Lukkarinen, Aleksi; Malmi, Lauri; Haaranen, Lassi

Event-driven Programming in Programming Education: A Mapping Review

Published in:
ACM Transactions on Computing Education

DOI:
10.1145/3423956

Published: 16/03/2021

Document Version
Peer reviewed version

Please cite the original version:
Lukkarinen, A., Malmi, L., & Haaranen, L. (2021). Event-driven Programming in Programming Education: A Mapping Review. *ACM Transactions on Computing Education*, *21*(1), [1]. https://doi.org/10.1145/3423956

ACM Reference format:
Aleksi Lukkarinen, Lauri Malmi, and Lassi Haaranen. 2021. Event-driven Programming in Programming Education: A Mapping Review. *ACM Trans. Comput. Educ.* 21, 1, Article 1 (March 2021), 31 pages.
https://doi.org/10.1145/3423956

Event-driven Programming in Programming Education

A Mapping Review

ALEKSI LUKKARINEN, LAURI MALMI, and LASSI HAARANEN

During the past two decades, *event-driven programming* (EDP) has emerged as a central and almost ubiquitous concept in modern software development: Graphical user interfaces are self-evident in most mobile and web-based applications, as well as in many embedded systems, and they are most often based on reacting to events. To facilitate both teaching practice and research in programming education, this mapping review seeks to give an overview of the related knowledge that is already available in conference papers and journal articles. Starting from early works of the 1990s, we identified 105 papers that address teaching practices, present learning resources, software tools or libraries to support learning, and empirical studies related to EDP. We summarize the publications, their main content, and findings. While most studies focus on bachelor's level education in universities, there has been substantial work in K–12 level, as well. Few courses address EDP as their main content—rather it is most often integrated with CS1, CS2, or computer graphics courses. The most common programming languages and environments addressed are Java, App Inventor, and Scratch. Moreover, very little of deliberate experimental scientific research has been carried out to explicitly address teaching and learning EDP. Consequently, while so-called experience reports, tool papers, and anecdotal evidence have been published, this theme offers a wide arena for empirical research in the future. At the end of the article, we suggest a number of directions for future research.

CCS Concepts: • **Social and professional topics** → **Computer science education**; • **Software and its engineering** → *Publish-subscribe / event-based architectures.*

Additional Key Words and Phrases: event-oriented, event-based, programming education, computer science education, K–12, CS0, CS1, CS1.5, CS2, CS3

1 INTRODUCTION

Event-drivenness has become a common characteristic of modern computer applications from productivity software and games to operating systems, embedded systems, remote services, and distributed systems. This trend has resulted in the concept of *event-driven*[1] *programming* (EDP). Naturally, the trend of growing importance of understanding EDP has been noticed in the discipline of

[1]Also terms *event-based* and *event-oriented* have been used.

About the authors: *Aleksi Lukkarinen*, Master of Science (Software Technology), male, ORCID iD 0000-0002-3827-6243, ResearcherID S-1254-2016, aleksi.lukkarinen@aalto.fi. *Lauri Malmi*, professor, male, ORCID iD 0000-0003-1064-796X, ResearcherID G-2346-2013, lauri.malmi@aalto.fi. *Lassi Haaranen*, Doctor of Science (Technology), male, ORCID iD 0000-0002-6500-6425, ResearcherID AAZ-3031-2020, lassi.haaranen@aalto.fi. All authors are affiliated with Aalto University, Department of Computer Science, P.O. Box 15400, FI-00076 Aalto, Finland. All authors declare no external funding and no conflicts of interest.

computing education long ago; this has prompted educators and researchers to express their opinions for the need to understand event-driven programming (EDP) [e.g., Bergin et al. 1999, Hansen and Fossum 2004, Bills and Biles 2005, Tchamgoue et al. 2011, Turbak et al. 2014]. Moreover, *Association for Computing Machinery* and *IEEE Computer Society* had included EDP already into their joint Computing Curricula 2001 as a core topic under the knowledge focus group *Programming Fundamentals*, and more recently, both Computing Science Curricula 2013 and Computer Engineering Curricula 2016 address EDP under several knowledge areas. For instance, the former includes knowledge unit *Fundamental Programming Concepts* under knowledge area *Software Development Fundamentals* with the following description:

> This knowledge unit builds the foundation for core concepts in the Programming Languages Knowledge Area, most notably in the paradigm-specific units: Object-Oriented Programming, Functional Programming, and Event-Driven & Reactive Programming.

As can be seen from the above description, these model curricula treat EDP as a programming paradigm. Krishnamurthi and Fisler [2019] challenge this position and remind that the concept of programming paradigm is not clearly defined. Procedural, object-oriented, and functional programming are fundamentally *organizational characteristics*, that is, they define the overarching method of arranging program code into manageable units. As Krishnamurthi and Fisler point out, event-drivenness, in contrast, is a *behavioural characteristic* and as such, orthogonal to organizational ones. This makes it inherently a different type of concept to teach and learn.

In introductory programming education, EDP usually relies on two central concepts. First, an *event* denotes an occurrence of something in hardware or software, or a data chunk that represents such an occurrence. Second, an *event handler* refers to a subprogram that is to be executed as a reaction to one or more kinds of events. Thus, compared to the traditional idea of sequentially executing a program from the beginning to the end, an event-driven program effectively waits endlessly for events and reacts to any events received as appropriate. As Woodworth and Dann [1999] note, this *inversion of control*, in the sense that the execution order significantly depends on external events, is a significant difference to the traditional approach.

While the need to educate students about EDP has been addressed, the educational practice has revealed many challenges related to teaching and learning EDP, starting from Bruce et al. [2001a], Bishop and Horspool [2004], El-Nasr and Smith [2006], Dabney et al. [2013], and Gordon et al. [2015] describing the concept as a whole as challenging or hard to understand. Some authors describe more specific obstacles. The most commonly reported one has been that relevant tools and libraries, such as *Java Abstract Window Toolkit* (AWT) initially and *Java Swing* later, are difficult to learn and use for novice programmers.[2] This concern is not directly related to the concept of EDP itself and it can be alleviated with an appropriate scaffolding: either wrapping existing class libraries inside simpler façades or creating simpler libraries from scratch.[3]

In addition to unfriendly programming environments, there are issues closer to EDP itself. Although event-drivenness is essentially about behaviour, using event handlers in organizing the program code is a common practice. This, for instance, requires the student to grasp the wholeness of the functionality that is distributed across the event handlers. It might also require building the program logic in a different way compared to a non-event-driven solution [e.g., Halland and

[2] For instance, Grissom [2000], Lambert and Osborne [2000], Bruce et al. [2001a], Alphonce and Ventura [2003], Bills and Biles [2005], Schaub [2009], Pauca and Guy [2012], Turbak et al. [2014], and Russo [2017].

[3] For instance, Roberts and Picard [1998], Wolz and Koffman [1999], Lambert and Osborne [2000], Bruce et al. [2001b], Rasala et al. [2001], Christensen and Caspersen [2002], Alphonce and Ventura [2003], Bishop and Horspool [2004], Murtagh [2007], and Russo [2017].

Malan 2003, Reges 2006, Salanci 2006], and novices might have challenges in figuring out how to design their code and how different parts of it work together [Jiang et al. 2004]. To properly understand and design event-driven programs, yet another need is an understanding of concepts that might not be covered until more advanced courses. For instance, a clear idea of origins and routing of events might require understanding of operating and windowing systems, device drivers, and hardware devices as embedded systems. Furthermore, designing event-driven programs successfully might require knowledge of design patterns, such as Observer, Model-View-Controller, and State. In addition, a multi-threaded programming environment might require understanding concerns and concepts related to concurrency.

The challenges related to event-driven programming (EDP) are nothing new. For instance, an early study from the beginning of the 2000s by Halland and Malan [2003] found out that several Computer Studies teachers, who had been teaching programming using *Pascal*, had problems with figuring out the layouts, including the main program and the place where the program would finally end, as well as the control flows of programs developed using *Delphi*. As the authors reflect, possible reasons for this include both event-drivenness as a concept and the different way of organizing the program code. As a result, it was necessary for the "teachers to develop new ways of thinking about – – how control works – –." Also, the experience of Bills and Biles [2005] from the end of the 1990s regarding teaching *Visual Basic* was that code "distributed across multiple objects too early in the learning process tended to confuse some students."

Halland and Malan as well as Bills and Biles observed also that some parts of the program might be completely hidden by the development environment, which makes it impossible to trace the program code from beginning to the end and thus hinders the understanding of the program. Jiang et al. [2004] worries that this kind of hiding might be a factor "impeding students' motivation to discover and understand – –." The hiding of code also contributes to the difficulty of debugging event-driven programs in addition to, for instance, loose coupling of event sources and listeners [Hansen and Fossum 2004]. Testing event-driven programs also has its own hindrances originating, for instance, the combinations of events and states [Hansen and Fossum 2004] as well as the tools needed to both send external events to the programs and observe their state. What is more, EDP has been associated with negative transfer effects observed between event-oriented and non-event-oriented programming environments [e.g., Woodworth and Dann 1999, Dingle and Zander 2000, Halland and Malan 2003, Ladd 2006]. Moreover, Halland and Malan express a concern that students who learn to program in an event-driven fashion do not develop some algorithmic skills that other programming students will have.

Despite that some challenges related to teaching and learning EDP are well known, there does not seem to be much deliberate experimental research towards alleviating them. For instance, research into the set of concepts that students should have to be able to successfully learn EDP seems not to exist. To find out the extent of missing EDP-related knowledge and to alleviate the problem as much as possible, we performed a *mapping review* of the topic. We systematically map out, categorize, and synthesize an overview of what the scientific community has published in scientific journals and conferences about teaching and learning EDP in programming education. In terms of Kitchenham, Budgen, and Brereton [2016, p. 34], we can classify this study as a *mapping study*:

> The goal of a mapping study is to survey the available knowledge about a topic. It is then possible to synthesize this by categorisation in order to identify where there are 'clusters' of studies that could perhaps form the basis of a fuller review, and also where there are 'gaps' indicating the need for more primary studies.

On the other hand, of the 14 review types of Grant and Booth [2009], this study conforms most closely a cross of a *systematized review* and a *mapping review/systematic map*. Of the above four

terms, we have chosen to use the *mapping review*, because it most clearly conveys the intent of our study: a *review* that *maps* literature.

In respect to the above study type, our study is exploratory in nature. Because it seems to be difficult[4] to find an exact general-level definition for event-driven programming (EDP), we treat it as a loosely-defined concept that the content of the publications relevant for the study will characterize. Furthermore, concerning publications that (1) are made through scientific journals and conferences and (2) are related to teaching and learning EDP in programming education, we pose the following broad research questions:

RQ 1	Who has written about teaching and learning *event-driven programming* (EDP), when, and where?
RQ 2	Which research methods have been used to study teaching and learning EDP?
RQ 3	In which educational contexts has EDP been discussed?
RQ 4	What kind of pedagogical approaches have been reported to support learning EDP?
RQ 5	What kind of software tools have been developed to support learning EDP?
RQ 6	What empirical results of learning EDP have been reported?

From our search, we specifically exclude publications that address technical implementations of EDP without educational context, such as technical research on programming languages or operating systems, despite that such tools or techniques might be used in educational contexts. Using the same principle, we exclude papers on design, implementation, or use of embedded systems with hardware-generated events, if there is no explicit connection to an educational context.

The compiled results support computer science teachers in several ways. The review provides references to a large number of studies where many types of pedagogical approaches and educational tools to support learning EDP have been developed in different educational contexts. These studies can inspire teachers to design better education for their own course contexts. Moreover, we combine information about empirical studies in which the effects of various pedagogical approaches and tools have been investigated, which can support selection of approaches. Finally, we hope to identify gaps in the literature where more research is needed.

This article is structured as follows. In Chapter 2, we discuss how we analyzed the identified pool of literature. This chapter is complemented by Appendix A, in which we detail our include/exclude criteria as well as both the searching and the first phase of screening. We continue in Chapter 3 by presenting the results, including summaries of various aspects of the literature as well as content analysis of the identified publications. For this chapter, in turn, we provide a few supplemental tables and a figure in Appendix B. Finally, we discuss the limitations of the study in Chapter 4 and summarize our main conclusions in Chapter 5. The above two appendices have been published as supplemental material alongside this article.

[4] Formulating an exact general-level definition for *event-driven programming* (EDP) seems to be challenging because of interpretational and demarcation issues, whose solutions seem to depend on the level of abstraction and the purpose of the definition. These issues include (1) definitions for the terms *programming* and *event*, (2) events as data structures (object/struct) versus sets of parameter values, (3) event-specific handlers versus one handler for all events, (4) real-time versus played-back events, (5) loop-based polling (e.g., an "event loop") of software-based queues versus hardware ports, (6) events that are hidden from the programmer, (7) synchronization structures, (8) ReactiveX libraries (http://reactivex.io/, accessed October 1, 2020), (9) interrupts, (10) programmable logic devices reacting to inbound signals, (11) calls of all methods, procedures, and functions as events, (12) the start of program execution as an event, (13) EDP as a programming paradigm (e.g., Krishnamurthi and Fisler [2019]), (14) EDP versus rule-based programming (e.g., de Paula et al. [2018]).

2 METHOD

In this section, we describe the process that we followed in this study. The process consists of five phases, as Figure 1 below presents. It began by the first author (AL) collecting the publications to be reviewed (*Searching*; § A.2) and screening them quickly (*Rapid Screening*; § A.3). Next, all authors participated in *Detailed Screening* (§ 2.1). After we confirmed the set of publications to be included in the study, we further collaborated in *Collecting & Analyzing Data* (§ 2.2) and *Reporting* our findings. The searching and screening process is summarized in Figure 2 and Figure 3 below. We present here the two first phases in a short summary. Their details are given in Appendix A in the supplemental material. We describe here the *Detailed Screening* phase accurately, because it is more relevant for the results of this review.

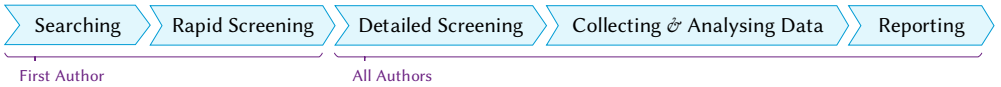

Fig. 1. Five phases of the review process followed in this study.

To find the included publications, we used eight search services: ACM Digital Library, IEEE Explore, Elsevier ScienceDirect, Springer Nature SpringerLink, Wiley Online Library, Taylor & Francis Online, Education Research Information Center, and Google Scholar (see Figure 2). The search expressions (see Table A2) were based on the following phrases: *event-driven programming*, *event-based programming*, *event-oriented programming*, *events-first programming*, and *event programming*. The search resulted in 858 documents, of which 643 were unique.

The unique search results were then screened based on, for instance, titles, content types, and prior knowledge of them to ensure that they follow our inclusion/exclusion criteria, which were: Publications (1) are accessible for free, (2) are published in journals or conference proceedings, (3) are written in English, (4) are not an introduction to some primary content (e.g., editorial to a journal special issue), (5) discuss events and programming in computer programming context, (6) are concerned with teaching and learning event-driven programming, and (7) are rated high enough in the publication rating system JUFO[5] used in Finland. For details of the criteria, see Appendix A in the supplemental material. During this process 327 search results were excluded and the final document pool for detailed screening included 316 publications.

2.1 Detailed Screening

After *Rapid Screening*, the *Detailed Screening* phase (Figure 3) commenced. During it, we evaluated search results by comparing their full content to the inclusion/exclusion criteria (Table A1). This comparison was not limited to abstracts, because in many cases, event-driven programming (EDP) was not a central concept for the document in question and thus was not mentioned in its abstract. As a part of this phase, the documents were routinely searched for the word *event* to quickly see, if an EDP-related term was being used in a relevant meaning and context. We categorized search results in three major categories:

- *Fully Included.* These 104 publications comprise content where EDP was researched or discussed in an educational context, that is, confirming the inclusion/exclusion criterion 6 about teaching and learning EDP. The publication could include, for example, a description of a pedagogical context where EDP had some clear role, present a software tool or other learning resource which addresses

[5] The JUFO rating systems has levels 0–3 and from the lowest level, we excluded publications, for which we did not find evidence of being peer reviewed. For more details, please see https://julkaisufoorumi.fi/en/publication-forum (accessed October 1, 2020) as well as § A.3 in the supplemental material.

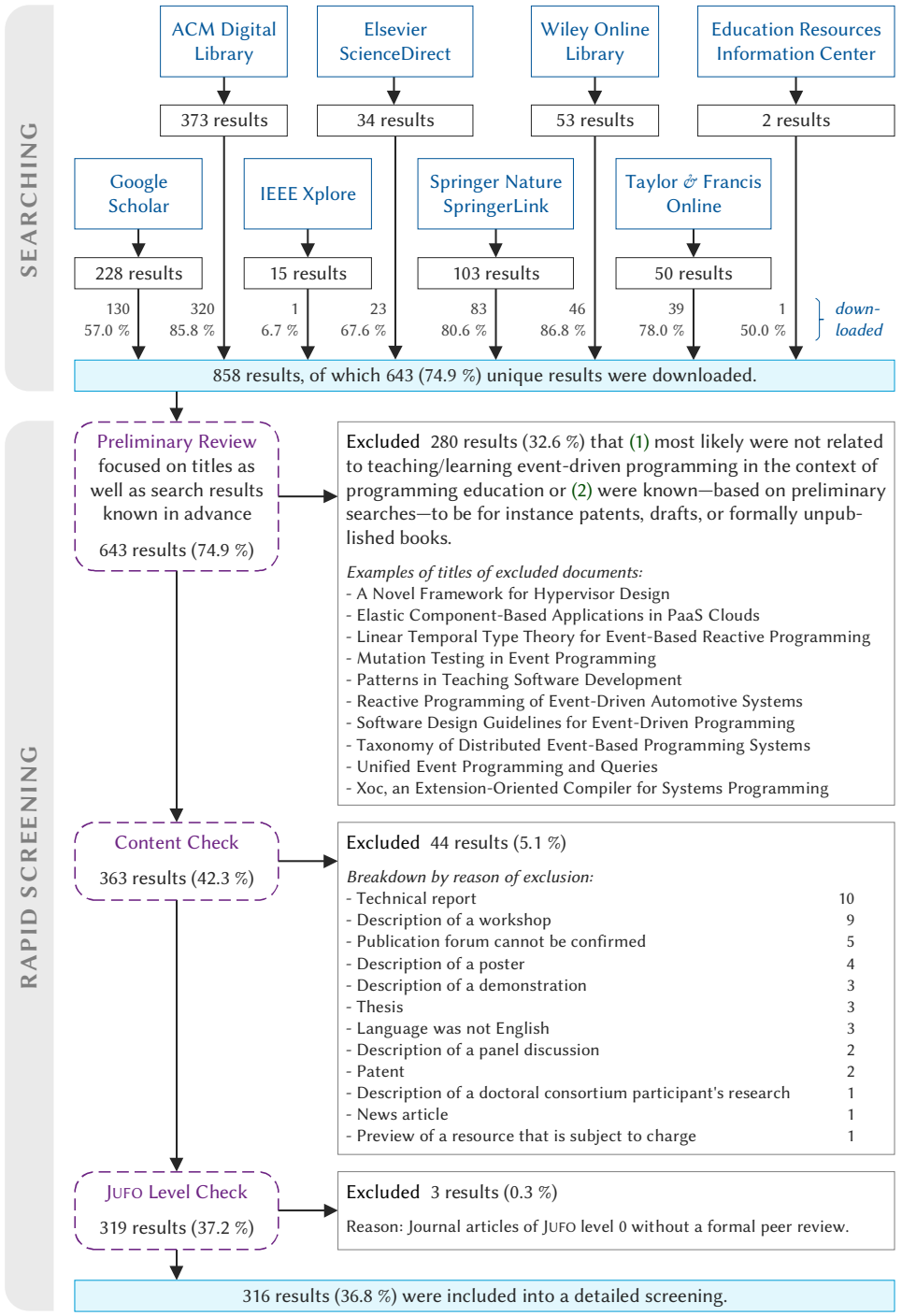

Fig. 2. An overview of the *Searching* and *Rapid Screening* phases. Percentages in parentheses are of all found results. These phases were performed by the first author. More details of the searches are provided in Table A2 and Table A3 in Appendix A. The process visualization continues in Figure 3 below.

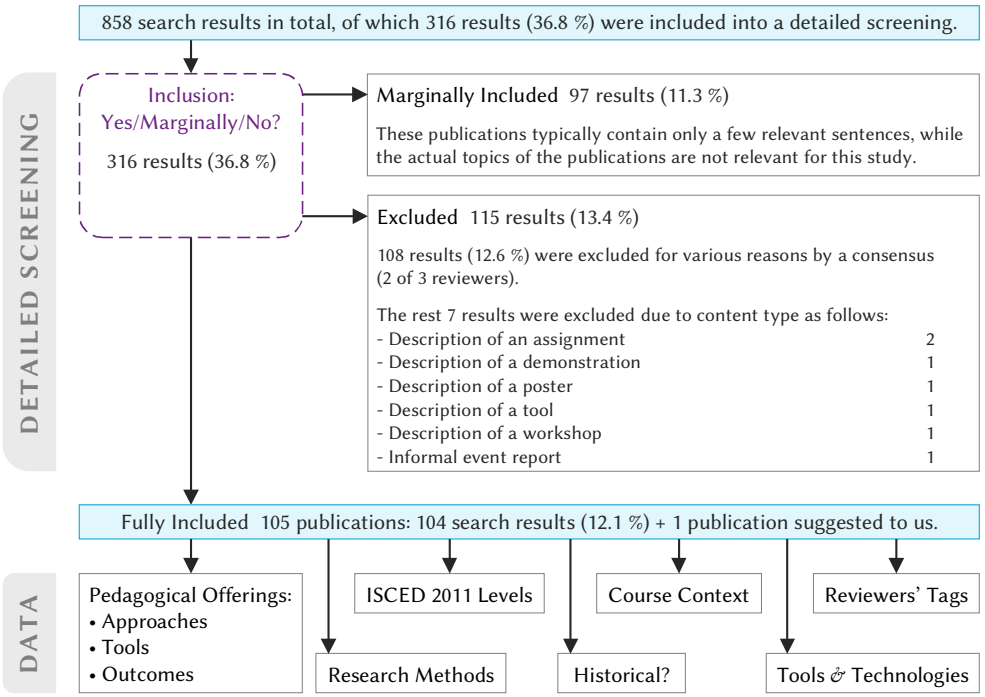

Fig. 3. An overview of the *Detailed Screening* and *Data Collection & Analysis* phases (continues from Figure 2). Percentages in parentheses are of all found results. These phases were performed by all authors together.

event-driven programming (EDP), or discuss challenges in learning EDP. We included publications that focused on some wider theme, if EDP was explicitly considered as a part of the bigger whole.

• *Marginally Included.* These 97 publications mentioned EDP in an educational context, while the main foci of them were clearly something quite different and not related to teaching or learning EDP. Most of them mentioned EDP briefly, without giving further details—typically in 1–2 sentences or as an item in a table as a topic of a curriculum or a course syllabus, or as a theme of an exercise. We therefore considered these publications less relevant for our survey, and do not discuss their content in more detail.

• *Excluded.* These 115 publications were not related to education at all or had clearly another focus, such as merely presenting technology that was not directed for education. We also excluded publications related to programming education, if closer investigation revealed that EDP was not considered in any relevant way.

Finally, we augmented the set of fully included publications by considering publications suggested by reviewers. After a similar detailed screening, we decided to add one paper, which was not captured among the initial search and was jointly deemed relevant for our survey.[6] This left us with the total of 105 publications to review.

All three authors participated in this phase, using the principle of majority vote. Two authors considered each publication independently and assigned them one of the three categories above. In case they agreed, the identified category became permanent. Otherwise, the third author read the publication followed by a joint discussion where a consensus of the category was reached.

[6] In the bibliography, we have included a number of publications, which were not accepted to the set of fully included publications but include some relevant aspects of EDP for discussion.

2.2 Collecting & Analyzing Data

After identifying the 105 fully included publications during *Detailed Screening*, we proceeded with the *Collecting & Analyzing Data* phase (Figure 1). We collected both substantial data (§ 2.2.1) and bibliometric data (§ 2.2.2), as we describe below.

2.2.1 Substantial Data

To collect the data necessary for analyzing the content of the fully included publications, we considered each publication regarding the following aspects (see Figure 3, *Data* section):

• *Pedagogical Offerings.* We identified content related to one or more of the three following subcategories: *Pedagogical approach* included a description of any form of teaching or learning practice or learning resource, regardless where event-driven programming (EDP) was involved as the main focus of research or as some minor content of the publication. *Tool* implied that the publication presented some software or hardware, which was specifically designed to support learning EDP. Typically, these were software frameworks that simplified learning some topic where EDP had a role, such as graphical user interface frameworks. *Learning outcomes* implied that the publication had some empirical content where learning results or changes in students' motivation, emotion, or attitude was analyzed.

• *Educational Levels.* We identified the target groups that had been in focus of the publication, if any, and used the *International Standard Classification of Education*[7] (ISCED 2011 version) to classify the educational levels of the target groups. Several levels were recorded, if necessary. In some publications, where no empirical data was involved, we concluded the level if it was implicit in the context; for instance, a CS1 course was categorized as bachelors' level education. For a small number of publications, such as literature studies or mere descriptions of educational software frameworks, we recorded educational level as N/A.

• *Research Methods.* The publications had a large variation of methodological approaches. To get a broad picture of them, we assigned each publication into one or more of the following categories:

- *Quantitative / Simple* publications presented some empirical data with simple descriptive statistics, such as summary tables, histograms, cross tabulations, simple correlations without any statistical testing.
- *Quantitative / Complex* publications applied some statistical testing to compare variables, or used exploratory statistical methods, such as clustering or factorial analysis. For these publications, we did not record *Quantitative / Simple* separately, even though they generally included such methods as well.
- *Qualitative / Simple* publications presented some collected qualitative data without presenting any clear analysis method. Typically these included quotes from student feedback or interviews, or observations on the target group, possibly presented with a simple categorization.
- *Qualitative / Complex* presented analysis of qualitative data with some clearly described analysis method and results which exceeded simple categorization, such as grounded theories or phenomenographical outcome spaces. For these publications, we did not record *Qualitative / Simple* separately, even though they generally included such actions/results as well.
- Publications which did not present any data were considered *Descriptive*. Typically, they presented some technical innovation, such as a new tool or software framework, compared

[7] See: http://uis.unesco.org/en/topic/international-standard-classification-education-isced, accessed October 1, 2020.

different technologies or presented new teaching methods or learning resources without any evaluation, expect possibly author's own reflections.

– *Literature reviews* presented and summarized a selected sample of literature.

• *Historicality.* We tagged a publication as historical if it concerned tools or technologies which are now practically obsolete, such as *Java Applets*.

• *Course Context and Theme.* Course names, such as CS1, CS2, or Computer Graphics, were reported as the course context, if any. In many cases, we also identified a specific *theme* which was in focus within the course context, such as developing games, building mobile apps, or working with media. However, not all papers had a course context or theme.

• *Tools & Technologies.* Teaching tools and technologies that the publication focused on, if any.

• *Reviewers' Tags.* Each reviewing author added some personal tags to identify any other possibly relevant aspects of the publication, which could be helpful in describing our findings. No formal categories were derived from the tags—they were considered simply auxiliary information.

We decided on pedagogical offerings, educational levels, research methods, and historicality based on the same majority vote principle as in *Detailed Screening* earlier. On the other hand, each reviewing author recorded course contexts, tools and technologies, individually because there were typically clearly stated in the papers.

2.2.2 Bibliometric Data

For each of the 105 fully included publications, we collected bibliometric data including authors, publication year, publication channel, page count, and citation count. Citations serve as approximations of the popularity and impact of a publication at some point of time. They are not universally exact, are changing over time, and can be made for very different reasons, not all of which relate to building on the ideas of the cited publications [Zupic and Čater 2015]. We did not, however, extend our survey to investigate the citations further. For our purposes, they served as orders of magnitude and can aid in high-level clustering and comparisons.

For this study, the citation counts represent mainly the viewpoint of *Scopus*[8] database on September 3, 2020 (in a few cases, other databases were also queried; see below). Scopus was selected as the source of citation data, because it might provide (1) more coverage than *Web of Science*[9] and (2) more accurate citation counts than Google Scholar [Falagas et al. 2007].

The interpretation of the results returned by Scopus was encumbered by some incoherence in them. For titles of some documents, Scopus returned several entries for seemingly unrelated publications; these entries were skipped, as were entries without a citation count. For these cases, we tried to collect the citation count from other sources, such as publishers' search engines and Google Scholar; this might have induced some distortion to the numbers recorded.

In cases when some of the content had been published several times, the citation counts of those publications were summed and the sum recorded. The same action was taken when Scopus returned several entries for a seemingly same publication but with some inaccuracies, such as one using the full name of a journal and the other one using its abbreviation, or both having the same publication but one of them having erroneous authors, and so on.

In the end, interpreting and cleaning the raw data as described above might have resulted in some ambiguity in the citation counts, and the authors acknowledge that some of the values might not be exact even in terms of Scopus' data. However, the possible error is usually quite small (such as 1–5 citations) and does not affect the big picture, especially as citation counts change over time.

[8] See: https://www.scopus.com/, accessed October 1, 2020.
[9] See: http://www.webofknowledge.com/, accessed October 1, 2020.

3 RESULTS AND DISCUSSION

We present the results of analysing the included publications in several sections, from several quantitative overviews to more detailed discussions in the order of our research questions (see Chapter 1). All sections are related to the fully included publications.

As an answer to our first research question, we present both bibliometrics (§ 3.1) as well as publication streams (§ 3.2). The latter are publication sets, whose members have common authors. We then briefly cover some historical publications (§ 3.3), which typically cover technologies, such as Java Applets, that are not being used any more. Next, we answer to RQ 2 by summarizing the research methods (§ 3.4) applied in the publications, and continue with RQ 3 by describing the educational contexts (§ 3.5), which include educational levels and participants (§ 3.5.1) as well as course contexts and themes (§ 3.5.2).

The more substantial results compiled from the analysis consist of answers to the rest of our research questions: Pedagogical approaches and observations (RQ 4, § 3.6), tools and frameworks (RQ 5, § 3.7), as well as learning outcomes (RQ 6, § 3.8), including any empirical results. Finally, we briefly discuss some ways to order learning content (§ 3.9). Details of the included publications are divided between the bibliography and Table B1 in the supplemental material.

3.1 Bibliometrics

As we stated earlier, the search process resulted in 858 search results, of which 643 unique ones (74.9 %) were downloaded for analysis. Of these 643, we excluded 83.8 % (539 results) from the study during the screening. This let us to include to the study 104 search results (16.2 %) and one additional publication that was recommended to us later—105 publications in total. We begin answering to our first research question below by summarizing bibliometrics describing the set of fully included publications (§ 3.1.1) as a whole, as well as the most cited publications (§ 3.1.2). Bibliometrics of the set of marginally included publications are omitted as being irrelevant. The answer to RQ 1 continues in § 3.2 with a discussion of publication streams.

3.1.1 Included Publications in General

We present the numbers of included publications by the publication year of their first versions (republications are omitted) in Figure 4. The publications were published during a continuous timespan of 26 years—from 1993 to 2018. As can be seen, the number of included publications per year starts to increase approximately at the change of the millennium; earlier the numbers were only marginal. From 1999 onward, there has been approximately 4.9 included publications per year. The peak years were 2014 and 2018 with 10 and 8 included publications, respectively.

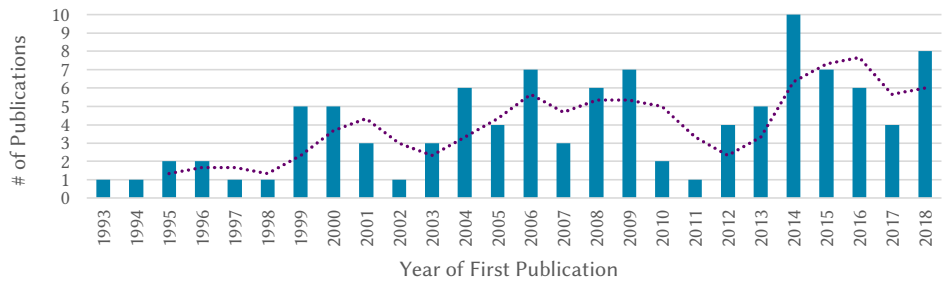

Fig. 4. Numbers of the included publications (see § 3.1.1) by the publication year of their first versions (republications are omitted). The dotted line represents a three-year moving average.

Figure 5 below presents the citation counts of these 105 included publications on a linear number line, and the 25 most cited publications are presented in Table B3. As can be seen, the publications form five clear groups. The first three groups consist of one publication, each: Two from Cooper, Dann, and Pausch [2000, 2003] with 297 and 248 citations, and one from El-Nasr and Smith [2006] with 123 citations. Furthermore, the fourth group comprises publications of both Seiter and Foreman [2013] as well as McCauley et al. [2008], which have been cited 82 and 73 times, respectively. Finally, the fifth group contains the other publications (95.2 %), which were cited up to 56 times.

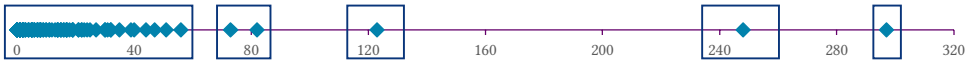

Fig. 5. Citation counts of the 105 included publications, as well as the five groups they form, on a linear number scale (see § 3.1.1).

The publications were authored by 252 unique authors, of which the most common one was *Kim B. Bruce*, who was an author in 6 publications (5.7 %). Both *Andrea P. Danyluk* and *Thomas P. Murtagh* appeared as an author in five publications, each, and both *Deborah A. Fields* and *Diana M. Franklin* in four publications, each. Finally, *Alexandria K. Hansen*, *Danielle B. Harlow*, *Wanda Dann*, and *Yasmin B. Kafai* participated in authoring of three publications, each. The three publications with most authors were those of Franklin et al. [2013] with 15 authors, Luxton-Reilly et al. [2017] with 10 authors, and Hodges et al. [2013] with 8 authors. 25 publications (23.8 %) had only one author, each, whereas 29 publications (27.6 %) had 2 authors, each.

The five most represented publication channels among the first versions of the 105 publications included to this study are presented in Table B2 (in Appendix B)—they cover 53.3 % (56) of the included publications. These channels are *Acm Sigcse Technical Symposium* (Sigcse) with 26 publications (24.8 %), *Innovation and Technology in Computer Science Education* conference (Iticse;[10] 11; 10.5 %), *Journal of Computing Sciences in Colleges* (Jcsc;[11] 11; 10.5 %), *Information Technology Education Conference* (Sigite;[12] 5; 4.8 %), and *Acm Transactions on Computing Education* (Toce;[13] 3; 2.9 %). The other channels were represented by 1–3 publications, each. All in all, of the 40 unique channels, 23 (57.5 %) were conferences and the rest 17 (42.5 %) journals. These channels represented 9 unique publishers, of which the most common one was by far Acm[14] with 65 publications (61.9 %). The second was *Consortium for Computing Sciences in Colleges* (Ccsc) with 12 publications (11.4 %), and the third was *Elsevier* with 7 publications (6.7 %). Finally, the *Lecture Notes in Computer Science*[15] (Lncs) publication series of *Springer*, as well as *Taylor & Francis*, had six publications, each.

A majority of the first published versions (75; 71.4 %) were quite short with 1–9 pages, but 18 (17.1 %) had 10–19 pages and the rest 12 (11.4 %) 20–29 pages. One likely reason for so many short papers being present is the page limits generally imposed on conference papers. Focusing on Jufo publication channel rating levels (see § A.3), most of the publications (77, 73.3 %) represented publication channels that have been classified on level 1 ("Basic"). The second most common rating

[10] Includes Iticse working group reports (Iticse-wgr).
[11] The proceedings of the conferences of *Consortium for Computing Sciences in Colleges* (http://www.ccsc.org/, accessed October 1, 2020) have been published in Jcsc, which is interpreted as a series of conference proceedings.
[12] Conferences Citc4 and Citc5 belong to the Sigite conference series and thus have been calculated as parts of it.
[13] During our timespan, Jeric was renamed to Toce. Despite the change in the journal profile, we calculate both as Toce.
[14] While Acm hosts the publications of Jcsc, their publisher is Ccsc. The same applies to Ace publications, which have been published by Australian Computer Society, Inc.
[15] See: https://www.springer.com/gp/computer-science/lncs, accessed October 1, 2020.

level was 0 with 18 (17.1 %) publications. Only ten publications (9.5 %) were from channels that represent the higher rating levels: Eight of them (7.6 %) were of level 2 ("Leading"), and the rest two (1.9 %) represented level 3 ("Highest"). Most of the rating levels (50; 47.6 %) were determined based on the publisher, and 39 publications (37.1 %) got their levels directly based on their publication channels. Four publications were assigned a rating level based on the Lecture Notes in Computer Science publication series, in which they have been published. Finally, in twelve cases (11.4 %) we had to use our own judgement to determine the appropriate level for the purposes of this study.

3.1.2 25 Most Cited Publications

Table B3 (in Appendix B) enumerates the 25 most-cited publications amongst the 105 fully included ones. They were written by 86 unique authors; 1–15 authors per publication. Only six authors were represented by more than one publication: *Deborah A. Fields*, *Kim B. Bruce*, *Randy Pausch*, *Stephen Cooper*, *Wanda Dann*, and *Yasmin B. Kafai* appeared as an author in two publications, each. The citation counts listed in the table are presented on a linear number line in Figure 5 along with the rest of the included publications. The groups are described above in § 3.1.1.

The table also lists information about the first published versions of these 25 publications. They were published during 15 individual years between 1997 and 2017 (that is, 6 individual years were not represented among these 25 publications). The peak was in 2013 with 4 publications; otherwise, there were 1–3 publications per year. The publications consist of 14 conference papers (56.0 %) and 11 journal articles (44.0 %), the former 1.3 times the latter. The publications were from 14 unique publication channels, of which the most common were the *Acm Sigcse Technical Symposium* (7 papers, 28.0 %), *Innovation and Technology in Computer Science Education* (including working group reports) conference (3 papers, 12.0 %), the *Journal of Computing Sciences in Colleges* (3 articles, 12.0 %), and *Acm Transactions on Computing Education* (2 articles, 8.0 %). The other ten channels were represented by one publication, each.

The most frequent publishers were the *Association for Computing Machinery* (16 publications, 64.0 %), *Taylor & Francis* (4 publications, 16.0 %), and *Consortium for Computing Sciences in Colleges* (3 publications, 12.0 %). *Australian Computer Society* and *Elsevier* were represented with one publication, each. Page counts varied between 5 and 29 as follows: 7 publications (28.0 %) had 20–29 pages, 5 (20.0 %) had 10–19 pages, and each of the rest 13 (52.0 %) consisted of 1–9 pages.

Concerning Jufo ratings, 17 (68.0 %) of the publications were on level 1 ("Basic"). In addition, level 3 ("Highest") had two publications (8.0 %), and levels 2 ("Leading") and 0 (does not meet the requirements of level 1) both had 3 publications (12.0 %) assigned to them. 8 (32.0 %) of these classifications were given based on the publication channel, while 14 (56.0 %) got the rating level assigned to their publisher. Moreover, the publication set included three publications that lacked an official rating and that we had to, for the purposes of this study, classify ourselves.

3.2 Publication Streams

We continue answering to our first research question by presenting four author-based publication streams that can be identified from the set of 105 fully included publications. They are publication sets of at least four members that have common authors. Figure 6 presents an overview of these streams; for more details, please see Figure B4 in the supplemental material.

The first publication stream includes publications authored by *Kim B. Bruce*, *Andrea P. Danyluk*, and *Thomas P. Murtagh*. At first, they argue [Bruce et al. 2001a] in favor of including event-driven programming (edp) into the introductory programming course from the beginning (an *events-first* approach) and present an *objects-first* course that uses their *ObjectDraw* library [Bruce et al. 2001b] as a scaffolding tool. Next, Bruce [2004] discusses the *objects-early* vs. *objects-late* debate and the above course, and presents three pedagogical directions to choose from. Finally, in the context of

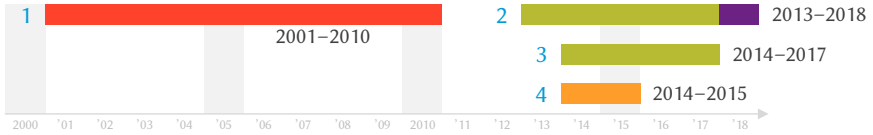

Fig. 6. An overview of the four author-based publication streams having at least four publications (see § 3.2). The first stream *(red)* is related mostly to ObjectDraw and Squint. The second and third streams are concerned with Scratch *(green)* and LaPlaya *(purple)*. Finally, the fourth stream *(orange)* is linked to App Inventor. More detail is shown in Figure B4 in Appendix B.

the same course, the authors argue why structural recursion should be taught earlier than arrays [Bruce et al. 2005] and how they introduce concurrency in the context of event-driven programming (EDP) [Bruce et al. 2010].

The second stream has four publications with *Diana M. Franklin* as the common author. First, Franklin et al. [2013] describe a two-week *Scratch*-based summer camp for middle school students and assess the shown competence of the students in areas such as EDP. In 2016, Franklin et al. investigate differences in state initialization between Scratch, *C*, and Java to make instruction to produce more transferable skills. The last two publications, Harlow et al. [2018] and Weintrop et al. [2018], discuss *LaPlaya* and *Kids Engaged in Learning Programming and Computer Science* (KELP-CS) curriculum that includes EDP. The former publication explores mismatches between LaPlaya and its users' abilities, continuing with the initialization theme of Franklin et al. [2016] above. The latter one presents such usage patterns of visual block-based languages that can be harmful from the viewpoint of later computer science education.

The third stream is formed by four Scratch-related publications that have *Deborah A. Fields* as the common author. In the first of them, Fields et al. [2014] identify programming profiles on the basis of analyzing Scratch projects regarding used programming concepts, level of participation, gender, and account age. The second publication [Fields et al. 2015] discusses high school students programming music videos as parts of collaborative communities, such as collectives of 3–6 students, the ongoing workshop, and the online Scratch community. Third, Fields et al. [2016] research children's learning trajectories by combining their social learning context with programming project snapshots. They especially examine EDP and parallelism as well as initialization, the latter adding to the work presented in Franklin et al. [2016] and Harlow et al. [2018] above. The last publication [Fields et al. 2017] is a journal article based on the first publication of this stream—a conference paper of the same authors from 2014.

The fourth stream consists of five publications related to *App Inventor*, published during two consecutive years. They do not have a single common author; however, they are mutually linked by *Mark Sherman*, *Andrey Soares*, and *Franklyn Turbak*, each appearing as an author in more than one of them. In the first publication, Soares [2014] reflects on topics that instructors should consider when teaching with App Inventor. During the same year, Turbak et al. discuss both the event model and some characteristics of App Inventor and event-drivenness. On the next year, Soares and Martin [2015] describe the results of a survey that seeks to find out students' opinions related to an *Android* application development course. Furthermore, Kim and Turbak [2015] investigate the usability of *map*, *filter*, *reduce*, and *sort* blocks in programming with App Inventor. Finally, Sherman and Martin [2015] suggest a rubric for assessing some aspects of computational thinking present in App Inventor projects.

3.3 Historical Publications

We categorized the 105 fully included publications according to whether the technology and the core content was still relevant to contemporary programming education. We considered 21 publications (20.0 %) to be historical; in other words, they focus on obsolete technologies such as Java Applets. Most of them were published in the 1990s or the early 2000s. In addition, we rated two of the publications [Bruce et al. 2005; Gestwicki and Sun 2008] to be semi-historical, that is, the technologies in focus are obsolete but the results or the analyses are still relevant. Finally, we considered the remaining 82 publications to contain research or discussion that is of interest to current practitioners and researches. However, it should be noted that the research contexts of these historical and semi-historical publications might still be relevant, even though the software presented might have to be adapted to contemporary technologies.

Software. Examples of what we considered as historical publications include discussions of obsolete platforms, such as *X Window System* [Pavlidis 1996], *Windows 3.0* [Szuecs 1996], and Java Applets [e.g., Christensen and Caspersen 2002; Tuttle 2001]. These (and similar) early works sought to resolve the same problem: While building interactive graphical applications and user interfaces motivated students to learn programming, the available tools were complex with a high learning curve. In 1996, Pavlidis starts his publication bluntly: "If you are teaching a graphics course with programming assignments, you will probably agree that using X for that purpose can be a troublesome experience." Similarly, Bruce et al. [2001b] write that "Our desire to use graphics extensively in the course made it clear that we would need to develop a set of classes to simplify the use of Java's graphics facilities. Java AWT does not provide an interface that is appealing for an introductory course." In the late 1990s and the early 2000s, many frameworks/libraries[16] were implemented to resolve this challenge of enabling students to start building graphical user interfaces (GUI) on CS1 or CS2 levels, or in introduction to computer graphics course.

In addition to GUIs, there are some other approaches worth of mentioning, as well. For instance, *XDB* [Arnow 1995] was an early library supporting learning distributed computing. ObjectDraw library simplified building graphics in Java. While it may be obsolete at the moment, Bruce et al. [2010] report about their 10-year long experience of giving a CS1 course, where event-driven programming and concurrency were taught from the very beginning. They describe their pedagogical planning and how the course proceeds week by week presenting concrete examples and assignments what students should solve. Finally, while *JavaScript* is currently the core web technology, it has gained its popularity during the last 10 years only. An interesting early publication by Ward and Smith [1998] proposes using JavaScript as the first language for multimedia students, presenting their course implementation and reflections.

Hardware. Another branch of early work concerns pedagogical solutions involving both programming and hardware. McNeill and Helm [1995] present a course on electronics and digital hardware, explaining various laboratory exercises. They discuss events in the context of an assignment addressing *hardware interrupts* and how these should be managed with software. A bit more recent publication of Tesser et al. [2000] presented a course directed for Integrated Science and Technology students. As an essential component, their approach includes building software which reads real instrument data using *LabWorks II Interface* device. They describe in some detail the laboratory exercises and their teaching approach, where teamworking and multi-science exercises are combined. This allows them to combine three goals, preparing students to gain CS1/CS2

[16] For instance: Alphonce and Ventura [2003]; Bruce and Danyluk [2004]; Bruce et al. [2001b]; Christensen and Caspersen [2002]; Lambert and Osborne [2000]; Lang and Saacks-Giguette [1999]; Leutenegger [2006]; Pavlidis [1996]; Roberts et al. [2006]; Szuecs [1996]; Tuttle [2001].

background, develop experience and skills appropriate to software engineering teams, and increase the number of students with good scientific problem solving skills.

Software Engineering. Finally, it is worthwhile to mention an early publication of Purtilo and Siegel [1994]. Related to software engineering education, it reports on several iterations of a large-scale team project course, where the goal was to implement a user-programmable event management environment.

3.4 Research Methods

We answer to our second research question by summarizing the research methods applied in the 105 fully included publications. 23 publications presented plain quantitative results—13 with simple methods and 10 with complex methods. Correspondingly, mere qualitative results were given in 12 publications—6 using simple and 6 complex methods. Mixed methods were used in 16 publications, as shown in Table 1 below.

Table 1. Numbers of publications using mixed methods (see § 3.4).

	Qualitative Simple	Qualitative Complex
Quantitative Simple	8	5
Quantitative Complex	1	2

Almost half of the publications (52) were plain descriptive—there was no clear data collection nor analysis. Many of these included authors' or teachers' reflections on the presented contribution, but we did not count this as data collection. Finally, one publication was a literature review.

3.5 Educational Contexts

Our third research question concerned the educational contexts of research. As an answer, the following two subsections discuss educational levels and participants (§ 3.5.1), as well as course contexts and themes (§ 3.5.2). In terms of them both, the contexts were very diverse.

3.5.1 Educational Levels and Participants

We present the educational levels using the ISCED categories (see § 2.2), which present educational contexts ranging from early-childhood education to doctoral education.

A clear majority of the publications (66 %, 69 publications) discussed event-driven programming at bachelor's level (ISCED level 6). K–12 level education was discussed in 38 % (40) of the publications, including 15 % (16) publications in high school level (ISCED 3), 12 % (13) in lower secondary level (ISCED 2), 10 % (10) in primary level (ISCED 1), and one publication in preschool level (ISCED 0). Seven publications discussed education at master's level, one at doctoral level, and one in vocational education. 10 publications did not relate to any specific level of education and they were classified as N/A. In some cases, where the level of education was not explicitly mentioned, we deduced it from the content. For example, an online course offered freely for all that covered topics similar to CS1 courses would be classified as ISCED level 6. A distribution of the ISCED levels is presented in Figure 7; for individual publications, please see Table B1 in the supplemental material.

Research had been carried out in multiple different courses with a large variety of course titles. To simplify the big picture, we categorized the courses under more generic titles. A large majority of publications (63 %, 66 publications) focused on a single (university level) course, and only four

publications discussed several different courses. School level education was reported in 27 publications (26 %). However, contrasting to university level education, these were not reported as clearly as courses. In 15 publications, the context was high school or school level in general and we list them as K–12 programming courses. If the context was clearly limited to primary education, we list them as K–6 programming courses (4) or pre-school context (1). In addition, a group of seven publications focused on school children, too, but in the context of various outreach activities instead of regular school education. Two publications focused on complementary education. The course contexts for each publication are included in Table B1.

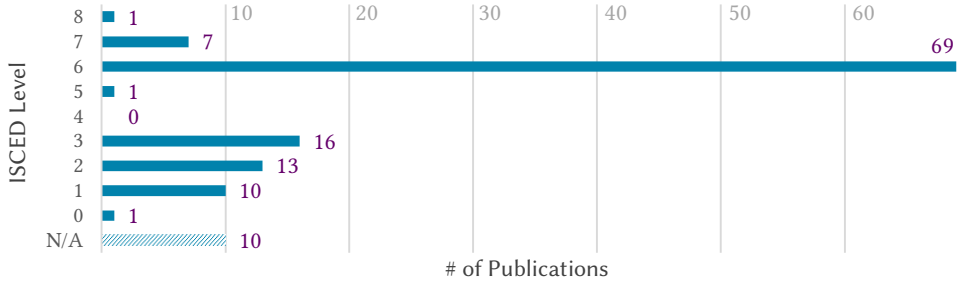

Fig. 7. Numbers of included publications per each ISCED level (see § 3.5.1). The levels are not exclusive: In many cases, one publication is included in several levels. The N/A group includes publications, for which we were not able to assign a clear educational level.

In addition, a few publications listed separately some additional characteristics of the target group, such as CS minors/non-CS-majors [Burns 2005; Kunkle and Allen 2016; Sherman and Martin 2015; Whitehead 2008], multimedia students [Ward and Smith 1998], students in a young researcher program [de Paula et al. 2018], MOOC (massive open online course) students [Tang et al. 2014], industrial design students [Alers and Hu 2009] or complementary education in arts, design and architecture [Sadi and Maes 2007].

3.5.2 *Course Contexts and Themes*

With *course context* we denote the name of the educational unit (e.g., CS1, CS2, or Computer Graphics) that was the target of research. On the other hand, often the more relevant context was the *theme* that was discussed in the publication within the course context, for instance, web programming in CS2 context, mobile programming in CS1, or games in K–12 programming. We present the course contexts first followed by themes; for individual publications, please see Table B1.

On the university level, the most common course context was introductory programming—typically called CS1—with 32 publications, followed by 7 contexts which discussed CS1 and CS2 education together. Computer graphics course was discussed in seven publications, where the main covered topics were user interaction with graphical objects and building graphical user interfaces. Four publications concerned specifically courses in mobile programming. The rest of the course contexts discussed various advanced programming courses after CS2 level, or some specific topics, which included artificial intelligence, programming languages, design patterns, event-driven programming, formal methods, game design, mechanical engineering & microprocessors, operating systems, network security, software engineering project, and web programming.

In five publications, there was no clear course context at all. These publications presented various tools, frameworks, or pedagogical approaches that support event-driven programming and could be implemented in many different courses. We list them with a title *CS curriculum*. There

were also two publications that discuss complementary education. Finally, two publications (a literature review and a publication discussing JavaScript style) had no course context at all.

Many themes were addressed within the target courses. The most popular themes discussed in the publications were mobile programming, typically using App Inventor, and graphics and GUI programming using various frameworks. Several publications addressed embedded programming and web programming, or working with multimedia or robots. Other themes included games, processing data from instruments, wireless sensors, parallelism, augmented or virtual reality, and design patterns. Note that not all publications had a clear theme in focus.

3.6 Pedagogical Approaches and Observations

This section is the first one of those that discuss more substantial results of this review. In it, we answer to our fourth research question about *pedagogical approaches*—the first category in our categorization of pedagogical offerings (see § 2.2.1). In 74 publications (70.5 %), *some* pedagogical approach was presented. We, however, found only a few publications that focus solely on teaching and learning event-driven programming (EDP). More commonly, EDP is discussed as a part of some wider context, such as graphical user interfaces (GUI), other kinds of computer graphics, mobile applications, games, or some other specific application areas.

Hansen and Fossum [2004], however, discuss an in-depth upper-division CS course in EDP. The course gives a comprehensive treatment of event-driven systems. It captures the significance of event-drivenness and integrates concepts from several computing fields: programming languages, operating systems, and software engineering.

Some authors explicitly mention the challenges of learning EDP. For instance, Jiang et al. [2004] discuss the challenges of teaching the first Visual Basic course. They write: "The biggest challenge to students is to figure out what to be placed inside the event handler of each control and the understanding of the relationship among different GUI objects. Students with little or no programming knowledge can experience difficulties in placing syntactically and semantically correct statements inside event handler procedures." The authors note that their students acknowledged that learning OOP helped them to learn GUI programming. Ladd [2006], whose publication focused on using games to motivate the *Net Generation* students to learn programming, also noted that "the change from having control of the game loop to an event-driven program is a real challenge." However, he continues that "many students want to have a 'pretty' program when they are done."

Hidalgo-Céspedes et al. [2018] discuss the effect of *metaphors and allegories* on learning programming, and present several examples, how abstract event-related concepts can be explained by using real life examples to concretize the concepts. On the other hand, when teaching high school students, Salanci [2006] contrasts the implementation of traditional programs and event-driven applications, and discusses what is easy to do in traditional programs but is not easy to transform to event-driven applications. Guo et al. [2016] followed children in six seventh-grade science classes, and present a case study of two children, who were asked to design a program for a hypothetical simulation of evolution. Comparing pre- and post-interviews they, interestingly, found that both children shifted from an event-based programming approach to a rule-based approach.

Contextualizing EDP—presumably making it more relevant (see, for instance, Lukkarinen and Sorva [2016])—has been a common theme in the investigated pedagogical approaches. From 2009 onwards, teaching EDP with web development was a common approach. de Raadt [2010] and Schaub [2009] investigated web development as a context for CS1 courses, while Stepp et al. [2009] researched the introduction to web programming as a bridge course between CS1 and CS2. Related to web development, Passier et al. [2014] created an approach and guidelines for students on how to create good code using JavaScript.

Wang [2014] presented a different context. He presented a novel course design, which adopted "the wireless sensor motes as a pedagogical tool to provide more effective Computer Science education on several complex topics, including the event-driven concurrency model in our Operating System course and the secure network communication protocol implementation in our Network Security course." He presented two hands-on projects based on *wireless sensors*, an event-driven concurrency programming project and a secure communication protocol implementation project, which were specifically designed for the two courses. He notes that "in the Operating System course, the wireless sensor event-driven programming project helps our students clearly understand the differences between the event-driven concurrency model and the multi-threading model."

Incorporating event-driven programming (EDP) into CS1 has been a common theme in research, as well. The publication stream based on work done by Bruce et al. focusing on EDP in CS1 was already discussed in § 3.2. However, already in 1999, Woodworth and Dann reported integrating event-driven approach in CS1, starting from console programs and moving onto GUI programming. A little later, Christensen and Caspersen [2002] published their *Presenter* framework as an alternative approach to teaching object-oriented and EDP concepts in CS1. Wicentowski and Newhall [2005] approached teaching CS1 and EDP through assignments centered around image manipulation. Their publication focused on describing their assignments but they also reported a positive response from students to this approach. Alphonce and Ventura [2003] approached teaching object-oriented CS1 using graphics utilizing the *NGP* library to avoid using Java Swing, which they considered too complex for beginners. They go on to discuss their approach and how it enhances teaching object-oriented concepts while taking care not to distract students from the core principles. Pecinovský et al. [2006] presents an approach to teach *design patterns* in CS1 and gives an example, where EDP is involved with the Observer design pattern.

Goldwasser and Letscher [2009] introduced *cs1graphics*—a tool for teaching graphics concepts in introductory programming using *Python*. They describe its use from the first day of programming to more intermediate topics, such as basic object-oriented programming (OOP), recursion, and events. Later on in the curriculum, Wolfe [1999] discussed various approaches they have for introductory graphics course for CS majors. Similarly, Pavlidis [1996] discussed their approach to teaching graphics using the X Window System. These discussions were considered historical in our review—so was the publication by Hitchner and Sowizral who, in 2000, discussed adapting the curricula to match changes in graphics programming, using *Java 3D* to build applets. Sung and Shirley [2004] argued that teaching graphics top–down, rather than bottom–up, works better for adult learners. They describe their course in computer graphics with this approach, depicting also an application architecture used as a model to guide event-processing in students' projects.

Mobile computing has been discussed in several publications. Research concerning the use of App Inventor was already discussed in § 3.2. Isolated from the presented publication stream, Perdikuri [2014] presented their experiences of using App Inventor and what concepts of EDP were involved in learning.

More general work in the mobile computing context include [Stuurman et al. 2014]. They used Android as an example and describe an anatomy of mobile apps, where EDP has an obvious role. Based on this anatomy, they present a selection of modeling techniques that can be applied to design mobile apps. On the other hand, Kumar [2018] presents how to incorporate mobile computing into the projects of two upper-level courses: Organization of Programming Languages and Artificial Intelligence. He discusses how to carry out the transition from OOP to event-driven Java and what aspects need to be covered. The publication continues to present further transition to using Android framework. After building the mobile programming competencies, he discusses projects in the upper level courses and evaluates student feedback.

Adding games or game-related topics to courses was also investigated in multiple publications. Gestwicki and Sun [2008] presented a case study of using games as a context for learning design patterns, arguing that not only they are motivating to students but they also naturally include many opportunities to use design patterns. Similarly, Dolgopolovas et al. [2018] as well as El-Nasr and Smith [2006] presented case studies related to motivation and use of games in programming classrooms. Dolgopolovas et al. [2018] introduced C programming through App Inventor with projects related to creating games, while El-Nasr and Smith [2006] use game modding (modifying existing games) as a way to teach principles of CS and event-driven programming.

Event-driven programming (EDP) has been discussed in various other educational contexts, as well. Liu [2008] wrote, how they used *Robocode* in teaching programming, and how EDP was included in tasks. Esteves et al. [2008], on the other hand, reported on experiences of using *Second Life* in programming education, and how also EDP was addressed. Chaytor and Leung [2003] presented examples how *.NET* technologies can be integrated in courses and covered several aspects of EDP. The publication by Goadrich [2014] demonstrated the integration of three different types of tangibles into the classroom: *Arduino* microcontrollers, Android phones, and *Sifteo Cubes*. They explained their laboratory assignments and lessons learned for each device. Finally, Tarkan and Sazawal [2009] presented event-driven development with the specification language *Alloy*, including a tutorial for Z-to-Alloy transition with concrete examples.

Some research projects focused on teaching non-majors CS skills that are related to EDP. Whitehead [2008] presented an experience report of a game design course for non-majors and reported a high rate of student satisfaction on the course. Burns [2005] proposed new courses for liberal arts students that would introduce the breadth of CS in one or two semesters while covering introductory programming. For teaching CS to multimedia students, Ward and Smith [1998] proposed using JavaScript as their first language. They report positive experiences from the course and describe their approach in detail. CS skills and programming are increasingly needed in mechanical engineering, as well. As early as in 1995, McNeill and Helm [1995] reported on a course in microprocessors targeted for mechanical engineering students, in which students implemented a capstone project where their software had to process various hardware interrupts.

Regarding assessment, Seiter and Foreman [2013] have developed a model called *Progression of Early Computational Thinking* (PECT) for primary school children. The model assesses event-driven programming in the forms of, for instance, *broadcasts*, user interface events, and parallelization.

3.7 Tools and Frameworks

The second category of our categorization of pedagogical offerings (see § 2.2.1) was *pedagogical tool*. As 34 publications (32.4 %) discussed *some* pedagogical tools, it was the second-largest pedagogical offering amongst the 105 fully included publications. In this section, we answer to our fifth research question by discussing teaching and learning tool offerings related to event-driven programming (EDP), many of which are targeted for CS1 or K–12 education. We begin, however, by summarizing the programming languages represented by the fully included publications.

Programming Languages. Out of the 105 publications we looked into, we identified 37 programming languages[17] that were central to the topic of the publication. 25 of these languages were mentioned in individual publications only. Ten languages (27.0 %), by contrast, were used in more

[17] The counts related to programming languages depend on interpretations, including: What is "essential" considering the publications? Should we count Visual Basic 6, Visual Basic .NET, and Visual Basic for Applications as different languages? How about Scratch, Snap!, and LaPlaya, or C and C++? Should SQL (the Structured Query Language) and specification languages such as Alloy and Z be included even if they might not be considered to be programming languages? Does a tiny bespoke language for a small chat application count? With different decisions, the results will vary.

than two publications—they have been summarized in Figure 8. The most popular language was Java with 30 publications (28.6 %), followed by App Inventor and C++ with 10 publications (9.5 %) each. Scratch took the fourth place with 8 publications (7.6 %). We note here that even though App Inventor and Scratch are clearly more popular now than Alice, which was used in 5 papers, they actually build much on the previous research on Alice. Alice, for example, introduced important ideas of drag&drop editor and storytelling with interaction into programming education. For more information on design principles of Alice, see Cooper [2010].

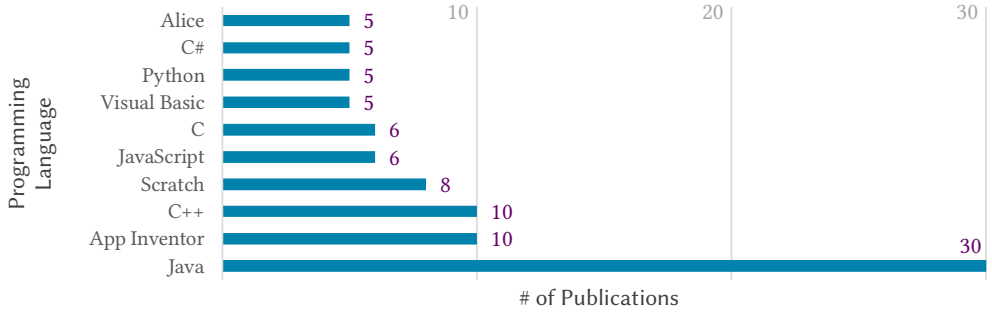

Fig. 8. Programming languages that were used in more than two publications.

Graphical User Interfaces / Computer Graphics. As we presented in the context of historical publications (§ 3.3), there is a significant branch of work that addresses the challenge on building interactive graphical user interfaces (GUI) and applications; we do not repeat it here. Also, the publication stream based on work done by Bruce et al. was discussed earlier (§ 3.2).

In CS1, Lambert and Osborne [2000] discussed a tool for CS1 to enable learning GUIs on the course using Java. Also using Java, Russo [2017] introduced *DoodlePad* aimed at CS1 to enable creation of graphics-related event-driven programming (EDP) assignments. Murtagh [2007] developed *Squint* library for Java, also for teaching EDP as well as concepts related to network applications.

Similar solutions have been later developed for Python, for instance, Goldwasser and Letscher [2009] present *cs1graphics*, "a new Python drawing package designed with pedagogy in mind. The intermediate and advanced lessons enabled by our graphics package involve flow of control, container classes, object-oriented principles, inheritance, recursion, and event-driven programming." The publication mainly focuses on teaching various Python features, but it also presents some example codes for teaching concepts of EDP.

A more advanced approach is *Views* [Bishop and Horspool 2004] tool, which "features an XML-based GUI description notation coupled with an engine that shields the programmer from much of the intricate complexity associated with events, listeners and handlers." The publication includes several examples how to use the framework in coding, including event handling and *callbacks*.

Robotics. Another group of tools, aimed at K–12 learners, were related to robots and teaching CS skills both in general and in event-driven programming. Magnenat et al. [2015] introduced *Thymio*, a robotics programming platform for CS education that also included augmented reality. Patten et al. [2000] introduced tangible interface for creating event-driven programs for controlling robots. For non-major university education, Alers and Hu [2009] presented *AdMoVeo*, a platform where the focus is on teaching programming concepts for designers.

3.8 Learning Outcomes

Here we address our sixth research question concerning empirical research on teaching event-driven programming. These results belong into the third category of our categorization of pedagogical offerings (see § 2.2.1)—*learning outcomes*. Unfortunately, the publications that we acknowledged to discuss learning outcomes and offer results about teaching or learning event-driven programming (EDP) formed only a subset of the 105 fully included publications (27, 25.7 %; see Table B1 in the supplemental material). Furthermore, there were virtually no experiment-based results that would indicate that one pedagogical method or tool is better than another and thus advice educators clearly and based on evidence.

Some authors describe characteristics and advantages/disadvantages of EDP and might list statistics about, for instance, the numbers of event handlers present in some sample of students' projects. Some authors present their personal opinions for or against EDP itself or some tool related to it. However, most of the publications treat learning of EDP as an automatic consequence of using some specific learning environment instead of experimentally researching into the actual learning outcomes. The publications that we briefly introduce below discuss teaching or learning EDP more than merely (1) claiming in passing that some method or tool caused students to learn and (2) giving some counts as statistics. The results are scattered tiny fragments, but the theme of visual block-based languages (VBBL) connects some of them together. Some of these publications we already introduced in § 3.2 and § 3.6.

Scratch and LaPlaya. Franklin et al. [2016] found out that some students initialized their projects in wrong types of events. Similarly, Harlow et al. [2018] report that some students "used unusual starting events but did not convey this to a user." Fields et al. [2016] analyze the usage of three event-driven concepts in projects of one student on the basis of both quantitative and qualitative data. Fields et al. [2017], in turn, analyze the numbers of block types used and identify a group of people who use EDP, in the form of broadcasts, more than others. Finally, Weintrop et al. [2018] remind us that teachers of students who might have been studying Scratch-like VBBLs earlier should acknowledge the inherent parallelism present in those languages and advice their students in sequential programming strategies accordingly. This is necessary, because with text-based languages in later education, parallelism is not necessarily introduced as an introductory topic, and some programming strategies that students learn with VBBLs, such as binding multiple actions to a single event, may not be applicable in non-parallel contexts.

App Inventor. Soares and Martin [2015] describe a survey of students who had participated in a course based on App Inventor. According to the survey, about 82.5 % of the students agreed or strongly agreed on a five-step Likert scale that developing mobile applications with App Inventor had "helped them to learn about" EDP. Naturally, this tells nothing about breadth or quality of the learning, and the students might not be knowledgeable enough to assess their own learning in the first place, yet. Dolgopolovas et al. [2018] discuss a method of exploiting (event-driven) App Inventor for motivating students to learn structured programming. They present a case study concerning one student, whose report was favourable for App Inventor.

Other Programming Languages. We described the finding of Guo et al. [2016] already in § 3.6. Liu [2008] discusses using Robocode programming game as a Java programming assignment and gives excerpts of students' reports, one of which demonstrates some understanding of EDP. Kunkle and Allen [2016] (see § 3.6) describe how the performance of students on a course using Visual Basic not only was poorer than that of students on Java and *C++* courses, but actually declined between pre- and post-tests. As an explanation for the decline, they suggest that "tenuous grasp of fundamental concepts in combination with the emphasis on designing interfaces and handling events might have caused the students to forget what they had learned in the previous course."

Finally, a historical study of Sheetz et al. [1997] found that in students' opinion, event-driven programming (EDP) "strongly affect the perceived difficulty of using *Smalltalk*, which in turn affects the difficulty of understanding object-oriented concepts."

Miscellaneous Publications. Two other studies are worth mentioning. First, Garner et al. [2005] categorized students' problems during laboratory sessions of an introductory programming course and had both event-driven programming and user interface programming as two separate problem categories. However, these two topics were introduced only in the very end of the course, so unfortunately the collected problem counts are not very representative compared to the other problem categories. Second, a study by Hidalgo-Céspedes et al. [2018] on the effects of teaching concepts related to event-driven programming using metaphors and allegories did not find significant differences between those two teaching approaches.

3.9 Orderings of Learning Content

In addition to covering varying aspects of teaching event-driven programming (EDP), a significant share of the publications in this study discuss pedagogical approaches regarding the order of the content to be taught on the introductory programming course. The approaches most frequently referred were objects-first and objects-early (e.g., Krishnamurthi and Fisler [2019]).

From the viewpoint of this review, the most interesting pedagogical approach is naturally the events-first approach; we already covered a significant thread of development related to it and its folloups [Bruce 2004; Bruce et al. 2001a,b, 2005, 2010] in the publication streams.

In addition to the above line of work, many other publications participate in the discussion. The workshop report published by Angster et al. [1999] discusses both teaching object-oriented programming (OOP; including objects-first) as well as more general aspects of teaching. Lambert and Osborne [2000] propose graphical user interfaces (GUIs) as being useful for demonstrating objects in an early phase of an introductory programming course, and provide their *BreezyGUI* library as the scaffolding tool. Cooper et al. [2003] discuss both the challenges of the objects-first approach and the development environment *Alice* as a means to overcome them.

Pecinovský et al. [2006] exemplify modifying objects-first teaching into *design-patterns-first*, and Utting [2006] compares the Standard and Micro Editions of Java 2 as teaching tools and suggests the latter for objects-first teaching. Mullins et al. [2009] discuss using Alice 2.0 as a programming environment in objects-first fashion, whereas Schaub [2009] highlights the complication of OOP not being suitable for small programs. Kunkle and Allen [2016] give examples of differing approaches (objects-early with *BlueJ*, objects-early with Alice, and *imperative-first* using Python), and report an experimental study on how the teaching approach and programming language affect learning. Finally, Dolgopolovas et al. [2018] consider with reservation the suitability of the objects-first approach for novices with the lowest programming skills.

In addition, two other approaches were mentioned. Burns [2005] present a *breath-first* course aimed for students not majoring in computer science. Guo et al. [2016] discuss how students' thinking changes to be based on rules instead of events after they had used a *code-first* programming environment called *Frog Pond* (see § 3.6).

4 LIMITATIONS

The aim of this study is to systematically map out, categorize, and synthesize an overview of what is published in scientific journals and conferences about teaching and learning event-driven programming (EDP). Despite our efforts, we recognize that our study has a number of limitations. First, there is no general consensus on the terminology used in the context of EDP. We used several different search terms to mitigate this problem. Second, EDP is relevant for many areas, not only in

software technology but also in embedded systems and various hardware-oriented applications. The scope of full-text search results was too large to be manageable as a whole and thus we had to limit the search to titles, abstracts, and keywords Some relevant work may have been missed. Still, it seems reasonable to assume that publications focusing on or addressing teaching and learning event-driven programming (EDP) mainly appear in the context of programming education, and advanced application areas are likely to be less relevant for our search. Third, we accepted publications written in English only—this may exclude some relevant work as well. However, we consider the risk here to be small.

Rapid Screening of the search results was carried out by one person only. Most inclusion/exclusion criteria can be clearly defined, causing little threat to the results. The only criterion where the risk on misjudgement is larger, concerns decisions on whether documents are concerned with teaching or learning EDP. Many publications focus on advancing software technology, for instance, presenting novel features in programming languages. These may have educational aspects as well, which may have been missed by one person alone. We deem this risk low, however, because abstracts generally report in some way, if the publication covers also some educational aspect.
Detailed Screening was carried out by three people using a majority vote: If two reviewers disagreed, a third researched considered the case and the final result was negotiated with consensus.

An obvious limitation is that we had to decide on a date limit for the search, after which we did not include more publications on the data pool. Thus our search pool presents the work that was available in the search services roughly till the middle of September 2018. A similar limitation concerns citation counts, as they increase over time.

5 DISCUSSION AND CONCLUSIONS

In this article, we have presented a mapping review concerning teaching and learning event-driven programming (EDP). To the review, we included 104 publications (12.1 %) from 858 search results, as well as one additional article that was suggested to us. These 105 publications concern a wide range of educational contexts, research methods, and pedagogical offerings. Most of the publications were related to some kinds of pedagogical approaches. Pedagogical tools and outcomes were discussed in a bit over one quarter of the publications, each.

It is clear that the majority of the research in this area has focused on university-level education. In the recent years, however, K–12 level education has been increasingly addressed; this is related to the increasing use of visual block-based programming languages. There has been some discussion and debate on the phase of the curriculum, in which EDP could or should be introduced and this discussed in intertwined with the discussion on how object-oriented programming should be introduced. Obviously, this depends on whether educational context is based on using traditional text-based languages, such as Java, Python, C++, and C#, or visual block-based programming languages, such as Scratch and App Inventor.

Noticeably, despite a number of personal reflections on the challenge of learning EDP, there was no consensus that learning the topic *per se* is difficult. More often the challenges were related to the actual tools available. Thus, a substantial share of research in university level education has focused on developing tools and frameworks that enable easier building of graphical user interfaces and interaction, thus hiding the inherent complexity of available (at the time) professional libraries and frameworks. This was discussed already in § 3.3. One important tool, not reported earlier in this work, was the `acm.graphics` package, initially published in ACM Java Task Force Final Report [Roberts et al. 2006]. This library obviously has had a wide impact on programming education, as it was particularly designed to simplify creating graphics applications for programming education in Java [e.g., Mertz et al. 2008]. It has been used as a model for graphics libraries for

other programming languages. For example, Roberts and Schwarz [2013] presented a corresponding graphics library for C/C++, and Goldwasser and Letscher [2009] presented the cs1graphics library for Python. On the other hand, for some educational settings, it has been considered too complex, and simpler solutions have been proposed, such the *Terminal Window Graphics* for C, proposed by Hovemeyer and Babcock [2009].

Much work of software tools and libraries has gotten dated due to some technologies, such as Java Applets, becoming obsolete. On the other hand, surprisingly little work has been published on learning event-driven programming (EDP) in web programming context with modern tools and languages, such as JavaScript.

EDP has became important in many software application areas. However, our major observation was that there is little empirical research that focuses on how students learn EDP and related concepts. Even these results are tiny and scattered fragments of information. Usually, EDP is discussed either in passing or on the side of some primary subject, such as a pedagogical tool or approach. Many publications present authors' personal reflections, anecdotal evidence of how students experienced some tool or educational setting, or treat learning EDP as an inherent property of being subjected to some tool or teaching approach. While such results can be valuable in communicating ideas, their generalizability is questionable at the best. Practical advice based on experimental research seems to be rare or missing altogether.

More generally, from the viewpoints of many subareas of Computing Education Research, EDP-related results did not exist among the publications we analyzed. We did not find any work related to students' misconceptions of EDP concepts, neither any systematic qualitative studies of their conceptions. Some qualitative studies focused on analyzing students' code and how certain structures, like initialization, had been used, while comprehensive analysis of students' thinking—when building event-driven programs—was missing. We therefore would like to see future research to evaluate teaching and learning EDP experimentally and more comprehensively. We propose some research ideas below.

• *Concepts and Skills.* What exact concepts and skills of EDP have really been learned, if any? How do students identify and understand central concepts or actions like events, event handlers or triggering events? This could imply, for example, interviewing students, allowing them to explain their own or an example program, and carrying out a phenomenographical or grounded theory analysis of the interview data.

• *Misconceptions.* What kind of misconceptions students may have on the basic concepts in EDP? Could we develop instruments to capture their understanding of EDP concepts, such that would augment the set of available instruments for investigating students' programming knowledge (see Margulieux et al. [2019]).

• *Notional Machines.* What would be an appropriate notional machine related to teaching EDP? How does it augment—or differ from—notional machines used for object-oriented programming?

• *Software Development.* How do students design, program, and debug their programs including EDP? What kind of practices and strategies they have? What difficulties, if any, do they encounter when designing and implementing programs with EDP? Here we could apply observation studies where they code exercise programs, or analyze their reflections on their programming projects possibly incorporated with interviews. This analyses could be used to build recommendations for teachers, how EDP should be taught in classes and presented in learning resources.

• *Methods and Tools.* How teaching methods and tools compare to each other in the results that can be achieved with them? How much a new teaching method or tool—or an alteration of an existing one—actually improves learning in different settings? Various research settings could be used here, to evaluate students' learning results, in terms of conceptual understanding or implementation of EDP. Are these result related to their other progress in learning programming?

• *Programming Languages.* How does teaching and learning event-driven programming (EDP) happen with different programming languages? What differences there are when learning EDP with Java, Python, C++, or some other languages? Obviously, block-based languages have a different learning process. Can students transfer what they have learned, for instance, in App Inventor or Scratch into implementing EDP using text-based languages?

• *Functional Languages.* How does learning EDP differ, if students use a functional language for building interactive applications?

• *Visualization.* How could software visualization support learning EDP-related concepts? What concepts should be visualized and how? What is the impact of using the visualization on students' understanding of EDP?

REFERENCES

Sjriek Alers and Jun Hu. 2009. AdMoVeo: A Robotic Platform for Teaching Creative Programming to Designers. In *Learning by Playing. Game-based Education System Design and Development, Edutainment 2009 (LNCS, vol. 5670)*, Maiga Chang, Rita Kuo, Kinshuk, Gwo-Dong Chen, and Michitaka Hirose (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, DE, 410–421. https://doi.org/10.1007/978-3-642-03364-3_49

Carl Alphonce and Phil Ventura. 2003. Using Graphics to Support the Teaching of Fundamental Object-Oriented Principles in CS1. In *Proc. 18th OOPSLA* (Anaheim, CA, USA). ACM, New York, NY, USA, 156–161. https://doi.org/10.1145/949344.949391

Erzsébet Angster, Joseph Bergin, and László Böszörményi. 1999. Introducing OO Design and Programming with Special Emphasis on Concrete Examples. In *Object-Oriented Technology ECOOP'99 Workshop Reader (LNCS, vol. 1743)*, Ana Moreira (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, DE, 338–361. https://doi.org/10.1007/11915355_17

David M. Arnow. 1995. XDP: A Simple Library for Teaching a Distributed Programming Module. In *Proc. 26th SIGCSE* (Nashville, TN, USA). ACM, New York, NY, USA, 82–86. https://doi.org/10.1145/199688.199732

Owen Astrachan, Kim B. Bruce, Elliot Koffman, Michael Kölling, and Stuart Reges. 2005. Resolved: Objects Early Has Failed. In *Proc. 36th SIGCSE* (St. Louis, MO, USA). ACM, New York, NY, USA, 451–452. https://doi.org/10.1145/1047344.1047359

Richard Austing, Robert D. Campbell, C. Fay Cover, Elizabeth K. Hawthorne, and Karl J. Klee. 2002. *Guidelines for Associate-Degree Programs in Computer Science.* Technical Report. ACM, New York, NY, USA. Retrieved Oct. 1, 2020 from https://dl.acm.org/doi/book/10.1145/2593245

Martin L. Barrett. 1993. A Hypertext Module for Teaching User Interface Design. In *Proc. 24th SIGCSE* (Indianapolis, IN, USA). ACM, New York, NY, USA, 107–111. https://doi.org/10.1145/169070.169359

Joseph Bergin, Amruth Kumar, Viera K. Proulx, Myles McNally, Alyce Faulstich Brady, David Mutchler, Stephen Hartley, Richard Rasala, Charles Kelemen, Rocky Ross, and Frank Klassner. 1999. Resources for Next Generation Introductory CS Courses: Report of the ITiCSE'99 Working Group on Resources for the Next Generation CS 1 Course. In *ITiCSE-WGR '99* (Cracow, PL). ACM, New York, NY, USA, 101–105. https://doi.org/10.1145/349316.571916

Dianne P. Bills and John A. Biles. 2005. The Role of Programming in IT. In *Proc. 6th SIGITE* (Newark, NJ, USA). ACM, New York, NY, USA, 43–49. https://doi.org/10.1145/1095714.1095727

Judith Bishop and Nigel Horspool. 2004. Developing Principles of GUI Programming Using Views. In *Proc. 35th SIGCSE* (Norfolk, VA, USA). ACM, New York, NY, USA, 373–377. https://doi.org/10.1145/971300.971429

Corey Brady, David Weintrop, Ken Gracey, Gabby Anton, and Uri Wilensky. 2015. The CCL-Parallax Programmable Badge: Learning with Low-Cost, Communicative Wearable Computers. In *Proc. 16th SIGITE* (Chicago, IL, USA). ACM, New York, NY, USA, 139–144. https://doi.org/10.1145/2808006.2808039

Kim B. Bruce. 2004. Controversy on How to Teach CS 1: A Discussion on the SIGCSE-Members Mailing List. In *Proc. ITiCSE-WGR* (Leeds, UK). ACM, New York, NY, USA, 29–34. https://doi.org/10.1145/1044550.1041652

Kim B. Bruce and Andrea P. Danyluk. 2004. Event-driven Programming Facilitates Learning Standard Programming Concepts. In *Companion 19th OOPSLA* (Vancouver, BC, CA). ACM, New York, NY, USA, 96–100. https://doi.org/10.1145/1028664.1028704

Kim B. Bruce, Andrea P. Danyluk, and Thomas P. Murtagh. 2001a. Event-Driven Programming Is Simple Enough for CS1. In *Proc. 6th ITiCSE* (Canterbury, UK). ACM, New York, NY, USA, 1–4. https://doi.org/10.1145/377435.377440

Kim B. Bruce, Andrea P. Danyluk, and Thomas P. Murtagh. 2001b. A Library to Support a Graphics-Based Object-First Approach to CS 1. In *Proc. 32nd SIGCSE* (Charlotte, NC, USA). ACM, New York, NY, USA, 6–10. https://doi.org/10.1145/364447.364527

Kim B. Bruce, Andrea P. Danyluk, and Thomas P. Murtagh. 2005. Why Structural Recursion Should Be Taught before Arrays in CS 1. In *Proc. 36th SIGCSE* (St. Louis, MO, USA). ACM, New York, NY, USA, 246–250.

https://doi.org/10.1145/1047344.1047430

Kim B. Bruce, Andrea P. Danyluk, and Thomas P. Murtagh. 2010. Introducing Concurrency in CS 1. In *Proc. 41st SIGCSE* (Milwaukee, WI, USA). ACM, New York, NY, USA, 224–228. https://doi.org/10.1145/1734263.1734341

Barry Burd, João Paulo Barros, Chris Johnson, Stan Kurkovsky, Arnold Rosenbloom, and Nikolai Tillman. 2012. Educating for Mobile Computing: Addressing the New Challenges. In *Proc. ITiCSE-WGR* (Haifa, IL). ACM, New York, NY, USA, 51–63. https://doi.org/10.1145/2426636.2426641

Brendan Burns. 2005. A New Approach to Computer Science in the Liberal Arts. *J. Comput. Sci. Coll.* 20, 5 (May 2005), 154–162. Retrieved Oct. 1, 2020 from http://dl.acm.org/doi/10.5555/1059888.1059943

Peter Chalk. 1999. Java in the Computing Curricula. *ACM SIGPLAN Notices* 34, 12 (Dec. 1999), 9–11. https://doi.org/10.1145/344283.344284

Louise Chaytor and Soleda Leung. 2003. How to Creatively Communicate Microsoft.NET Technologies in the IT Curriculum. In *Proc. 4th CITC (CITC4)* (Lafayette, IN, USA). ACM, New York, NY, USA, 168–173. https://doi.org/10.1145/947121.947160

Henrik Bærbak Christensen and Michael E. Caspersen. 2002. Frameworks in CS1 – A Different Way of Introducing Event-Driven Programming. In *Proc. 7th ITiCSE* (Aarhus, DK). ACM, New York, NY, USA, 75–79. https://doi.org/10.1145/544414.544438

Robert Coe, Michael Waring, Larry V. Hedges, and James Arthur (Eds.). 2017. *Research Methods and Methodologies in Education* (2nd ed.). SAGE Publications, London, UK.

Stephen Cooper. 2010. The Design of Alice. *ACM Trans. Comput. Educ.* 10, 4, Article 15 (Nov. 2010), 16 pages. https://doi.org/10.1145/1868358.1868362

Stephen Cooper, Wanda Dann, and Randy Pausch. 2000. Alice: A 3-D Tool for Introductory Programming Concepts. *J. Comput. Sci. Coll.* 15, 5 (2000), 108–117. Retrieved Oct. 1, 2020 from https://dl.acm.org/doi/10.5555/364133.364161

Stephen Cooper, Wanda Dann, and Randy Pausch. 2003. Teaching Objects-First in Introductory Computer Science. In *Proc. 34th SIGCSE* (Reno, NV, USA). ACM, New York, NY, USA, 191–195. https://doi.org/10.1145/611892.611966

Matthew H. Dabney, Brian C. Dean, and Tom Rogers. 2013. No Sensor Left Behind: Enriching Computing Education with Mobile Devices. In *Proc. 44th SIGCSE* (Denver, CO, USA). ACM, New York, NY, USA, 627–632. https://doi.org/10.1145/2445196.2445378

James Devine, Joe Finney, Peli de Halleux, Michał Moskal, Thomas Ball, and Steve Hodges. 2019. MakeCode and CODAL: Intuitive and Efficient Embedded Systems Programming for Education. *J. Syst. Archit.* 98 (2019), 468–483. https://doi.org/10.1016/j.sysarc.2019.05.005

Jill P. Dimond, Sarita Yardi, and Mark Guzdial. 2009. Mediating Programming through Chat for the OLPC. In *CHI EA* (Boston, MA, USA). ACM, New York, NY, USA, 4465–4470. https://doi.org/10.1145/1520340.1520684

Adair Dingle and Carol Zander. 2000. Assessing the Ripple Effect of CS1 Language Choice. *J. Comput. Sci. Coll.* 16, 2 (Oct. 2000), 85–93. Retrieved Oct. 1, 2020 from https://dl.acm.org/doi/abs/10.5555/369279.369331

Vladimiras Dolgopolovas, Tatjana Jevsikova, and Valentina Dagienė. 2018. From Android games to coding in C—An approach to motivate novice engineering students to learn programming: A case study. *Comput. Appl. Eng. Educ.* 26, 1 (2018), 75–90. https://doi.org/10.1002/cae.21862

Magy Seif El-Nasr and Brian K. Smith. 2006. Learning through Game Modding. *Comput. Entertain.* 4, 1, Article 7 (Jan. 2006), 20 pages. https://doi.org/10.1145/1111293.1111301

Micaela Esteves, Ricardo Antunes, Benjamim Fonseca, Leonel Morgado, and Paulo Martins. 2008. Using Second Life in Programming's Communities of Practice. In *Groupware: Design, Implementation, and Use, CRIWG 2008 (LNCS, vol. 5411)*, Robert O. Briggs, Pedro Antunes, Gert-Jan de Vreede, and Aaron S. Read (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, DE, 99–106. https://doi.org/10.1007/978-3-540-92831-7_9

Matthew E. Falagas, Eleni I. Pitsouni, George A. Malietzis, and Georgios Pappas. 2007. Comparison of PubMed, Scopus, Web of Science, and Google Scholar: strengths and weaknesses. *FASEB J.* 22, 2 (Sept. 2007), 338–342. https://doi.org/10.1096/fj.07-9492LSF

Annette Feng, Mark Gardner, and Wu-Chun Feng. 2017. Parallel Programming with Pictures is a Snap! *J. Parallel and Distrib. Comput.* 105 (2017), 150–162. https://doi.org/10.1016/j.jpdc.2017.01.018

Deborah A. Fields, Michael Giang, and Yasmin B. Kafai. 2014. Programming in the Wild: Trends in Youth Computational Participation in the Online Scratch Community. In *Proc. 9th WiPSCE* (Berlin, DE). ACM, New York, NY, USA, 2–11. https://doi.org/10.1145/2670757.2670768

Deborah A. Fields, Yasmin B. Kafai, and Michael T. Giang. 2017. Youth Computational Participation in the Wild: Understanding Experience and Equity in Participating and Programming in the Online Scratch Community. *ACM Trans. Comput. Educ.* 17, 3, Article 15 (Aug. 2017), 22 pages. https://doi.org/10.1145/3123815

Deborah A. Fields, Lisa Quirke, Janell Amely, and Jason Maughan. 2016. Combining Big Data and Thick Data Analyses for Understanding Youth Learning Trajectories in a Summer Coding Camp. In *Proc. 47th SIGCSE* (Memphis, TN, USA). ACM, New York, NY, USA, 150–155. https://doi.org/10.1145/2839509.2844631

Deborah A. Fields, Veena Vasudevan, and Yasmin B. Kafai. 2015. The Programmers' Collective: Fostering Participatory Culture by Making Music Videos in a High School Scratch Coding Workshop. *Interact. Learn. Environ.* 23, 5 (2015), 613–633. https://doi.org/10.1080/10494820.2015.1065892

Alice E. Fischer. 2011. That's Neat – How Do I Do It?: Demonstration. *J. Comput. Sci. Coll.* 26, 6 (June 2011), 61–63. Retrieved Oct. 1, 2020 from http://dl.acm.org/doi/10.5555/1968521.1968537

Diana M. Franklin, Phillip Conrad, Bryce Boe, Katy Nilsen, Charlotte Hill, Michelle Len, Greg Dreschler, Gerardo Aldana, Paulo Almeida-Tanaka, Brynn Kiefer, Chelsea Laird, Felicia Lopez, Christine Pham, Jessica Suarez, and Robert Waite. 2013. Assessment of Computer Science Learning in a Scratch-based Outreach Program. In *Proc. 44th SIGCSE* (Denver, CO, USA). ACM, New York, NY, USA, 371–376. https://doi.org/10.1145/2445196.2445304

Diana M. Franklin, Charlotte Hill, Hilary A. Dwyer, Alexandria K. Hansen, Ashley O. Iveland, and Danielle B. Harlow. 2016. Initialization in Scratch: Seeking Knowledge Transfer. In *Proc. 47th SIGCSE* (Memphis, TN, USA). ACM, New York, NY, USA, 217–222. https://doi.org/10.1145/2839509.2844569

Ilenia Fronza, Nabil El Ioini, and Luis Corral. 2016. Teaching Software Design Engineering Across the K–12 Curriculum: Using Visual Thinking and Computational Thinking. In *Proc. 17th SIGITE* (Boston, MA, USA). ACM, New York, NY, USA, 97–101. https://doi.org/10.1145/2978192.2978220

Sandy Garner, Patricia Haden, and Anthony V. Robins. 2005. My Program Is Correct But It Doesn't Run: A Preliminary Investigation of Novice Programmers' Problems. In *Proc. 7th ACE (Vol. 42)* (Newcastle, NSW, AU). ACS, Darlinghurst, NSW, AU, 173–180. Retrieved Oct. 1, 2020 from https://dl.acm.org/doi/10.5555/1082424.1082446

Vaidas Gasiunas, Lucas Satabin, Mira Mezini, Angel Núñez, and Jacques Noyé. 2011. EScala: Modular Event-Driven Object Interactions in Scala. In *Proc. 10th AOSD* (Porto de Galinhas, BR). ACM, New York, NY, USA, 227–240. https://doi.org/10.1145/1960275.1960303

Paul Gestwicki and Fu-Shing Sun. 2008. Teaching Design Patterns through Computer Game Development. *J. Educ. Resour. Comput.* 8, 1, Article 2 (March 2008), 22 pages. https://doi.org/10.1145/1348713.1348715

Mark Goadrich. 2014. Incorporating Tangible Computing Devices into CS1. *J. Comput. Sci. Coll.* 29, 5 (May 2014), 23–31. Retrieved Oct. 1, 2020 from http://dl.acm.org/doi/10.5555/2600623.2600627

Michael H. Goldwasser and David Letscher. 2009. A Graphics Package for the First Day and Beyond. In *Proc. 40th SIGCSE* (Chattanooga, TN, USA). ACM, New York, NY, USA, 206–210. https://doi.org/10.1145/1508865.1508945

Michal Gordon, Eileen Rivera, Edith Ackermann, and Cynthia Breazeal. 2015. Designing a Relational Social Robot Toolkit for Preschool Children to Explore Computational Concepts. In *Proc. 14th IDC* (Boston, MA, USA). ACM, New York, NY, USA, 355–358. https://doi.org/10.1145/2771839.2771915

Maria J. Grant and Andrew Booth. 2009. A typology of reviews: an analysis of 14 review types and associated methodologies. *Health Info Libr J* 26, 2 (June 2009), 91–108. https://doi.org/10.1111/j.1471-1842.2009.00848.x

Chris Gregg, Luther Tychonievich, James Cohoon, and Kim Hazelwood. 2012. EcoSim: A Language and Experience Teaching Parallel Programming in Elementary School. In *Proc. 43rd SIGCSE* (Raleigh, NC, USA). ACM, New York, NY, USA, 51–56. https://doi.org/10.1145/2157136.2157155

Scott Grissom. 2000. A Pedagogical Framework for Introducing Java I/O in CS1. *SIGCSE Bull.* 32, 4 (Dec. 2000), 57–59. https://doi.org/10.1145/369295.369326

Shuchi Grover and Roy Pea. 2013. Using a Discourse-Intensive Pedagogy and Android's App Inventor for Introducing Computational Concepts to Middle School Students. In *Proc. 44th SIGCSE* (Denver, CO, USA). ACM, New York, NY, USA, 723–728. https://doi.org/10.1145/2445196.2445404

Yu Guo, Aditi Wagh, Corey Brady, Sharona T. Levy, Michael S. Horn, and Uri Wilensky. 2016. Frogs to Think with: Improving Students' Computational Thinking and Understanding of Evolution in a Code-First Learning Environment. In *Proc. 15th IDC* (Manchester, UK). ACM, New York, NY, USA, 246–254. https://doi.org/10.1145/2930674.2930724

Ken Halland and Katherine Malan. 2003. Reflections by Teachers Learning to Program. In *SAICSIT 2003* (Johannesburg, ZA). South African Institute for Computer Scientists and Information Technologists, ZA, 165–172. Retrieved Oct. 1, 2020 from https://dl.acm.org/doi/abs/10.5555/954014.954032

Stuart Hansen and Timothy Fossum. 2004. Events Not Equal to GUIs. In *Proc. 35th SIGCSE* (Norfolk, VA, USA). ACM, New York, NY, USA, 378–381. https://doi.org/10.1145/971300.971430

Danielle B. Harlow, Hilary A. Dwyer, Alexandria K. Hansen, Ashley O. Iveland, and Diana M. Franklin. 2018. Ecological Design-Based Research for Computer Science Education: Affordances and Effectivities for Elementary School Students. *Cogn. Instr.* 36, 3 (2018), 224–246. https://doi.org/10.1080/07370008.2018.1475390

Jeisson Hidalgo-Céspedes, Gabriela Marín-Raventós, Vladimir Lara-Villagrán, and Luis Villalobos-Fernández. 2018. Effects of oral metaphors and allegories on programming problem solving. *Comput. Appl. Eng. Educ.* 26, 4 (2018), 852–871. https://doi.org/10.1002/cae.21927

Lewis E. Hitchner and Henry A. Sowizral. 2000. Adapting computer graphics curricula to changes in graphics. *Comput. Graph.* 24, 2 (2000), 283–288. https://doi.org/10.1016/S0097-8493(99)00162-4

Steve Hodges, James Scott, Sue Sentance, Colin Miller, Nicolas Villar, Scarlet Schwiderski-Grosche, Kerry Hammil, and Steven Johnston. 2013. .NET Gadgeteer: A New Platform for K–12 Computer Science Education. In *Proc. 44th SIGCSE* (Denver, CO, USA). ACM, New York, NY, USA, 391–396. https://doi.org/10.1145/2445196.2445315

David Hovemeyer and David Babcock. 2009. Using Terminal Window Graphics in CS1. *J. Comput. Sci. Coll.* 24, 3 (Jan. 2009), 151–158. Retrieved Oct. 1, 2020 from https://dl.acm.org/doi/10.5555/1409873.1409902

Keyuan Jiang, John Maniotes, and Reza Kamali. 2004. A Different Approach of Teaching Introductory Visual Basic Course. In *Proc. 5th CITC (CITC5)* (Salt Lake City, UT, USA). ACM, New York, NY, USA, 219–223. https://doi.org/10.1145/1029533.1029586

Ricardo Jiménez-Peris, Sami Khuri, and Marta Patiño-Martínez. 1999. Adding Breadth to CS1 and CS2 Courses through Visual and Interactive Programming Projects. In *Proc. 30th SIGCSE* (New Orleans, LA, USA). ACM, New York, NY, USA, 252–256. https://doi.org/10.1145/299649.299774

Joint Task Group on Computer Engineering Curricula, ACM, and IEEE-CS. 2016. *Computer Engineering Curricula 2016: Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering.* ACM, New York, NY, USA. Retrieved Oct. 1, 2020 from https://www.acm.org/education/curricula-recommendations

Annie Kelly, Lila Finch, Monica Bolles, and R. Benjamin Shapiro. 2018. BlockyTalky: New programmable tools to enable students' learning networks. *Int. J. Child Comput. Interact.* 18 (2018), 8–18. https://doi.org/10.1016/j.ijcci.2018.03.004

Soojin Kim and Franklyn Turbak. 2015. Adapting higher-order list operators for blocks programming. In *Proc. VL/HCC (IEEE Cat. No. CFP 15060-ART)* (Atlanta, GA, USA). IEEE, Piscataway, NJ, USA, 213–217. https://doi.org/10.1109/VLHCC.2015.7357219

Barbara Ann Kitchenham, David Budgen, and Pearl Brereton. 2016. *Evidence-Based Software Engineering and Systematic Reviews.* CRC Press (Taylor & Francis Group), Boca Raton, FL, USA.

Shriram Krishnamurthi and Kathi Fisler. 2019. Programming Paradigms and Beyond. In *The Cambridge Handbook of Computing Education Research*, Sally A. Fincher and Anthony V. Robins (Eds.). Cambridge University Press, Cambridge, UK, Chapter 13, 377–413. https://doi.org/10.1017/9781108654555.014

Amruth N. Kumar. 2018. Collateral Learning of Mobile Computing: An Experience Report. In *Proc. 23rd ITiCSE* (Larnaca, CY). ACM, New York, NY, USA, 27–32. https://doi.org/10.1145/3197091.3197106

Wanda M. Kunkle and Robert B. Allen. 2016. The Impact of Different Teaching Approaches and Languages on Student Learning of Introductory Programming Concepts. *ACM Trans. Comput. Educ.* 16, 1, Article 3 (Jan. 2016), 26 pages. https://doi.org/10.1145/2785807

Brian C. Ladd. 2006. The Curse of Monkey Island: Holding the Attention of Students Weaned on Computer Games. *J. Comput. Sci. Coll.* 21, 6 (June 2006), 162–174. Retrieved Oct. 1, 2020 from http://dl.acm.org/doi/10.5555/1127442.1127464

Ken Lambert and Martin Osborne. 2000. Easy, Realistic GUIs with Java in CS1. In *Proc. 2nd Annual CCSC Northwestern Conference* (Oregon Graduate Institute, Beaverton, OR, USA). Consortium for Computing Sciences in Colleges, USA, 209–215. Retrieved Oct. 1, 2020 from http://dl.acm.org/doi/10.5555/369279.369358

R. Raymond Lang and Marguerite Saacks-Giguette. 1999. Introducing High School Students to Event Driven Programming. In *Proc. 29th FIE (IEEE Cat. No. 99CH37011)* (San Juan, PR, USA), Vol. 1. IEEE, Piscataway, NJ, USA, 11B5/9–11B514. https://doi.org/10.1109/FIE.1999.839233

Scott T. Leutenegger. 2006. A CS1 to CS2 Bridge Class Using 2D Game Programming. *J. Comput. Sci. Coll.* 21, 5 (May 2006), 76–83. Retrieved Oct. 1, 2020 from http://dl.acm.org/doi/10.5555/1127351.1127366

Peter L. Liu. 2008. Using Open-Source Robocode as a Java Programming Assignment. *ACM SIGCSE Bull.* 40, 4 (Nov. 2008), 63–67. https://doi.org/10.1145/1473195.1473222

Aleksi Lukkarinen and Juha Sorva. 2016. Classifying the Tools of Contextualized Programming Education and Forms of Media Computation. In *Proc. 16th Koli Calling* (Koli National Park, Lieksa, FI). ACM, New York, NY, USA, 51–60. https://doi.org/10.1145/2999541.2999551

Andrew Luxton-Reilly, Brett A. Becker, Yingjun Cao, Roger McDermott, Claudio Mirolo, Andreas Mühling, Andrew Petersen, Kate Sanders, Simon, and Jacqueline Whalley. 2017. Developing Assessments to Determine Mastery of Programming Fundamentals. In *Proc. ITiCSE-WGR* (Bologna, IT). ACM, New York, NY, USA, 47–69. https://doi.org/10.1145/3174781.3174784

Stéphane Magnenat, Morderchai Ben-Ari, Severin Klinger, and Robert W. Sumner. 2015. Enhancing Robot Programming with Visual Feedback and Augmented Reality. In *Proc. 20th ITiCSE* (Vilnius, LT). ACM, New York, NY, USA, 153–158. https://doi.org/10.1145/2729094.2742585

Lauren Margulieux, Tuba Ayer Ketenci, and Adrienne Decker. 2019. Review of measurements used in computing education research and suggestions for increasing standardization. *Comput. Sci. Educ.* 29, 1 (2019), 49–78. https://doi.org/10.1080/08993408.2018.1562145

Lee McCauley, Jim Greer, David Mills, Jeff Robertson, and Allen Thomas. 2006. Teaching Objects First Using Lego Robots: A Tri-P-LETS Initiative. *J. Comput. Sci. Coll.* 21, 5 (May 2006), 183–185. Retrieved Oct. 1, 2020 from

http://dl.acm.org/doi/10.5555/1127351.1127384

Renée McCauley, Sue Fitzgerald, Gary Lewandowski, Laurie Murphy, Beth Simon, Lynda Thomas, and Carol Zander. 2008. Debugging: a review of the literature from an educational perspective. *Comput. Sci. Educ.* 18, 2 (2008), 67–92. https://doi.org/10.1080/08993400802114581

Myles F. McNally. 1998. Using Image Processing as a Laboratory Theme in CS1 and CS2 (Poster). In *Proc. 3rd ITiCSE* (Dublin City Univ., IE). ACM, New York, NY, USA, 292. https://doi.org/10.1145/282991.283622

Stephen R. McNeill and Jeffrey D. Helm. 1995. A required mechanical engineering course in microprocessors. *Mechatronics* 5, 7 (1995), 763–774. https://doi.org/10.1016/0957-4158(95)00047-9

Andrew Mertz, William Slough, and Nancy Van Cleave. 2008. Using the ACM Java Libraries in CS 1. *J. Comput. Sci. Coll.* 24, 1 (Oct. 2008), 16–26. Retrieved Oct. 1, 2020 from https://dl.acm.org/doi/10.5555/1409763.1409769

Paul M. Mullins and Michael Conlon. 2008. Engaging Students in Programming Fundamentals Using Alice 2.0. In *Proc. 9th SIGITE* (Cincinnati, OH, USA). ACM, New York, NY, USA, 81–88. https://doi.org/10.1145/1414558.1414584

Paul M. Mullins, Deborah Whitfield, and Michael Conlon. 2009. Using Alice 2.0 as a First Language. *J. Comput. Sci. Coll.* 24, 3 (Jan. 2009), 136–143. Retrieved Oct. 1, 2020 from http://dl.acm.org/doi/10.5555/1409873.1409900

Jogesh K. Muppala. 2011. Teaching Embedded Software Concepts Using Android. In *Proc. 6th WESE* (Taipei, TW). ACM, New York, NY, USA, 32–37. https://doi.org/10.1145/2077370.2077375

Thomas P. Murtagh. 2007. Squint: Barely Visible Library Support for CS1. In *Proc. 38th SIGCSE* (Covington, KY, USA). ACM, New York, NY, USA, 526–530. https://doi.org/10.1145/1227310.1227489

Gwen Nugent, Leen-Kiat Soh, Ashok Samal, and Jeff Lang. 2006. A Placement Test for Computer Science: Design, Implementation, and Analysis. *Comput. Sci. Educ.* 16, 1 (2006), 19–36. https://doi.org/10.1080/13803610500298094

Jackie O'Kelly and J. Paul Gibson. 2006. RoboCode & Problem-Based Learning: A Non-Prescriptive Approach to Teaching Programming. *ACM SIGCSE Bull.* 38, 3 (June 2006), 217–221. https://doi.org/10.1145/1140123.1140182

Samatios J. Papadakis and Vasileios Orfanakis. 2015. Οι εκπαιδευτικοί από καταναλωτές σε δημιουργούς ψηφιακού περιεχομένου μέσω του προγραμματιστικού περιβάλλοντος App Inventor for Android. In *Proc. 1ο Πανελλήνιο Επιστημονικό Συνέδριο, Πρακτικά Β Τόμος* (Heraklion, Crete, GR). Institute of Humanities and Social Sciences (I.A.K.E.), Heraklion, Crete, GR, 666–674. Retrieved Oct. 1, 2020 from https://iake.weebly.com/praktika2015.html

Sofia Papavlasopoulou, Kshitij Sharma, Michail Giannakos, and Letizia Jaccheri. 2017. Using Eye-Tracking to Unveil Differences Between Kids and Teens in Coding Activities. In *Proc. 2017 IDC* (Stanford, CA, USA). ACM, New York, NY, USA, 171–181. https://doi.org/10.1145/3078072.3079740

Harrie J. M. Passier, Sylvia Stuurman, and Harold Pootjes. 2014. Beautiful JavaScript: How to Guide Students to Create Good and Elegant Code. In *Proc. CSERC* (Berlin, DE). ACM, New York, NY, USA, 65–76. https://doi.org/10.1145/2691352.2691358

James Patten, Laurie Griffith, and Hiroshi Ishii. 2000. A Tangible Interface for Controlling Robotic Toys. In *CHI EA* (The Hague, NL). ACM, New York, NY, USA, 277–278. https://doi.org/10.1145/633292.633454

Victor Paul Pauca and Richard T. Guy. 2012. Mobile Apps for the Greater Good: A Socially Relevant Approach to Software Engineering. In *Proc. 43rd SIGCSE* (Raleigh, NC, USA). Association for Computing Machinery, New York, NY, USA, 535–540. https://doi.org/10.1145/2157136.2157291

Bruno Henrique de Paula, Andrew Burn, Richard Noss, and José Armando Valente. 2018. Playing Beowulf: Bridging Computational Thinking, Arts and Literature through Game-Making. *Int. J. Child Comput. Interact.* 16 (2018), 39–46. https://doi.org/10.1016/j.ijcci.2017.11.003

Theo Pavlidis. 1996. How to Teach Graphics Using X (and Live to Tell About It). *ACM SIGGRAPH Comput. Graph.* 30, 3 (Aug. 1996), 41–42. https://doi.org/10.1145/232301.232336

Rudolf Pecinovský, Jarmila Pavlíčková, and Luboš Pavlíček. 2006. Let's Modify the Objects-First Approach into Design-Patterns-First. In *Proc. 11th ITiCSE* (Bologna, IT). ACM, New York, NY, USA, 188–192. https://doi.org/10.1145/1140124.1140175

Katerina Perdikuri. 2014. Students' Experiences from the Use of MIT App Inventor in Classroom. In *Proc. 18th PCI* (Athens, GR). ACM, New York, NY, USA, Article 41, 6 pages. https://doi.org/10.1145/2645791.2645835

James Purtilo and Stan Siegel. 1994. Experiences with CCB-directed projects in the classroom. In *Software Engineering Education, CSEE 1994 (LNCS, vol. 750)*, Jorge L. Díaz-Herrera (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, DE, 283–302. https://doi.org/10.1007/BFb0017621

Michael de Raadt. 2010. Introductory Programming in a Web Context. In *Proc. 12th ACE (Vol. 103)* (Brisbane, AU). ACS, Darlinghurst, AU, 79–86. Retrieved Oct. 1, 2020 from http://dl.acm.org/doi/10.5555/1862219.1862232

Richard Rasala, Jeff Raab, and Viera K. Proulx. 2001. Java Power Tools: Model Software for Teaching Object-Oriented Design. In *Proc. 32nd SIGCSE* (Charlotte, NC, USA). ACM, New York, NY, USA, 297–301. https://doi.org/10.1145/364447.364606

Stuart Reges. 2006. Back to Basics in CS1 and CS2. In *Proc. 37th SIGCSE* (Houston, TX, USA). ACM, New York, NY, USA, 293–297. https://doi.org/10.1145/1121341.1121432

Derek Riley. 2012. Using Mobile Phone Programming to Teach Java and Advanced Programming to Computer Scientists. In *Proc. 43rd SIGCSE* (Raleigh, NC, USA). ACM, New York, NY, USA, 541–546. https://doi.org/10.1145/2157136.2157292

Eric Roberts, Kim Bruce, James H. Cross, Robb Cutler, Scott Grissom, Karl Klee, Susan Rodger, Fran Trees, Ian Utting, and Frank Yellin. 2006. The ACM Java Task Force: Final Report. In *Proc. 37th SIGCSE* (Houston, TX, USA). ACM, New York, NY, USA, 131–132. https://doi.org/10.1145/1121341.1121384

Eric Roberts and Antoine Picard. 1998. Designing a Java Graphics Library for CS 1. In *Proc. 6th ITiCSE* (Dublin City Univ., IE). ACM, New York, NY, USA, 213–218. https://doi.org/10.1145/282991.283129

Eric Roberts and Keith Schwarz. 2013. A Portable Graphics Library for Introductory CS. In *Proc. 18th ITiCSE* (Canterbury, England, UK). ACM, New York, NY, USA, 153–158. https://doi.org/10.1145/2462476.2465590

José María Rodríguez Corral, Antón Civit Balcells, Arturo Morgado Estévez, Gabriel Jiménez Moreno, and María José Ferreiro Ramos. 2014. A game-based approach to the teaching of object-oriented programming languages. *Comput. Educ.* 73 (2014), 83–92. https://doi.org/10.1016/j.compedu.2013.12.013

Mark F. Russo. 2017. DoodlePad: Next-gen Event-driven Programming for CS1. *J. Comput. Sci. Coll.* 32, 4 (April 2017), 99–105. Retrieved Oct. 1, 2020 from http://dl.acm.org/doi/10.5555/3055338.3055356

Sajid Sadi and Pattie Maes. 2007. subTextile: Reduced event-oriented programming system for sensate actuated materials. In *Proc. VL/HCC (IEEE Comput. Soc. Order No. P2987)* (Coeur d'Alene, ID, USA). IEEE, Piscataway, NJ, USA, 171–174. https://doi.org/10.1109/VLHCC.2007.37

Lubomir Salanci. 2006. Object-Oriented Programming at Upper Secondary School for Advanced Students. In *Informatics Education – The Bridge between Using and Understanding Computers, ISSEP 2006 (LNCS, vol. 4226)*, Roland T. Mittermeir (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, DE, 179–188. https://doi.org/10.1007/11915355_17

Christopher Scaffidi and Christopher Chambers. 2012. Skill Progression Demonstrated by Users in the Scratch Animation Environment. *Int. J. Hum. Comput. Interact.* 28, 6 (2012), 383–398. https://doi.org/10.1080/10447318.2011.595621

Stephen Schaub. 2009. Teaching CS1 with Web Applications and Test-driven Development. *ACM SIGCSE Bull.* 41, 2 (June 2009), 113–117. https://doi.org/10.1145/1595453.1595487

Barry Edward Schliesmann, Christopher William Weiland, and Timothy Wise. 2004. System and Method for Event Driven Programming. US Patent Application 20040205698, Filed December 28, 2001.

Linda Seiter and Brendan Foreman. 2013. Modeling the Learning Progressions of Computational Thinking of Primary Grade Students. In *Proc. 9th ICER* (San Diego, CA, USA). ACM, New York, NY, USA, 59–66. https://doi.org/10.1145/2493394.2493403

Steven D. Sheetz, Gretchen Irwin, David P. Tegarden, H. James Nelson, and David E. Monarchi. 1997. Exploring the Difficulties of Learning Object-Oriented Techniques. *J. Manag. Inf. Syst.* 14, 2 (1997), 103–131. https://doi.org/10.1080/07421222.1997.11518167

Mark Sherman. 2014. User Models of Reasoning and Understanding in App Inventor. In *Proc. 10th ICER* (Glasgow, Scotland, UK). ACM, New York, NY, USA, 171–172. https://doi.org/10.1145/2632320.2632340

Mark Sherman and Fred Martin. 2015. The Assessment of Mobile Computational Thinking. *J. Comput. Sci. Coll.* 30, 6 (June 2015), 53–59. Retrieved Oct. 1 2020 from http://dl.acm.org/doi/10.5555/2753024.2753037

Marian Sherwood. 1996. Access in Seattle. *3C ON-LINE* 3, 1 (Jan. 1996), 7–8. https://doi.org/10.1145/218806.218809

Andrey Soares. 2014. Reflections on Teaching App Inventor for Non-Beginner Programmers: Issues, Challenges and Opportunities. *Inf. Syst. Educ. J.* 12, 4 (July 2014), 56–65. Retrieved Oct. 1, 2020 from http://isedj.org/2014-12/n4/ISEDJv12n4p56.html

Andrey Soares and Nancy L. Martin. 2015. Teaching Non-Beginner Programmers with App Inventor: Survey Results and Implications. *Inf. Syst. Educ. J.* 13, 5 (Sept. 2015), 24–36. Retrieved Oct. 1, 2020 from http://isedj.org/2015-13/n5/ISEDJv13n5p24.html

Mary Stepp, Jessica Miller, and Victoria Kirst. 2009. A "CS 1.5" Introduction to Web Programming. In *Proc. 40th SIGCSE* (Chattanooga, TN, USA). ACM, New York, NY, USA, 121–125. https://doi.org/10.1145/1508865.1508908

Sylvia Stuurman, Bernard E. van Gastel, and Harrie J. M. Passier. 2014. The Design of Mobile Apps: What and How to Teach?. In *Proc. 4th CSERC* (Berlin, DE). ACM, New York, NY, USA, 93–100. https://doi.org/10.1145/2691352.2691360

Kelvin Sung and Peter Shirley. 2004. A top-down approach to teaching introductory computer graphics. *Comput. Graph.* 28, 3 (June 2004), 383–391. https://doi.org/10.1016/j.cag.2004.03.005

Kelvin Sung, Peter Shirley, and Becky Reed Rosenberg. 2007. Experiencing Aspects of Games Programming in an Introductory Computer Graphics Class. In *Proc. 38th SIGCSE* (Covington, KY, USA). ACM, New York, NY, USA, 249–253. https://doi.org/10.1145/1227310.1227400

Kim Sungkyung and Kim Sangchul. 2018. Middle-School Programming Classes Utilizing Game Creation and the Analysis of their Educational Outcomes. *Journal of Korea Game Society* 18, 3 (June 2018), 49–60. https://doi.org/10.7583/JKGS.2018.18.3.49

Laszlo Szuecs. 1996. Creating Windows Applications Using Borland's OWL Classes. In *Proc. 27th SIGCSE* (Philadelphia, PA, USA). ACM, New York, NY, USA, 145–149. https://doi.org/10.1145/236452.236528

Nour Tabet, Huda Gedawy, Hanan Alshikhabobakr, and Saquib Razak. 2016. From Alice to Python. Introducing Text-based Programming in Middle Schools. In *Proc. ITiCSE* (Arequipa, PE). ACM, New York, NY, USA, 124–129. https://doi.org/10.1145/2899415.2899462

Terry Tang, Scott Rixner, and Joe Warren. 2014. An Environment for Learning Interactive Programming. In *Proc. 45th SIGCSE* (Atlanta, GA, USA). ACM, New York, NY, USA, 671–676. https://doi.org/10.1145/2538862.2538908

Sureyya Tarkan and Vibha Sazawal. 2009. Chief Chefs of Z to Alloy: Using a Kitchen Example to Teach Alloy with Z. In *Teaching Formal Methods, TFM 2009 (LNCS, vol. 5846)*, Jeremy Gibbons and José Nuno Oliveira (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, DE, 72–91. https://doi.org/10.1007/978-3-642-04912-5_6

Guy Martin Tchamgoue, Ok-Kyoon Ha, Kyong-Hoon Kim, and Yong-Kee Jun. 2011. A Taxonomy of Concurrency Bugs in Event-Driven Programs. In *Software Engineering, Business Continuity, and Education, ASEA 2011 (CCIS, vol. 257)*, Tai-hoon Kim, Hojjat Adeli, Haeng-kon Kim, Heau-jo Kang, Kyung Jung Kim, Akingbehin Kiumi, and Byeong-Ho Kang (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, DE, 437–450. https://doi.org/10.1007/978-3-642-27207-3_48

Herbert Tesser, Hisham Al-Haddad, and Gary Anderson. 2000. Instrumentation: A Multi-science Integrated Sequence. In *Proc. 31st SIGCSE* (Austin, TX, USA). ACM, New York, NY, USA, 232–236. https://doi.org/10.1145/330908.331861

The Joint Task Force on Computing Curricula, ACM, and IEEE-CS. 2013. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science.* ACM, New York, NY, USA. Retrieved Oct. 1, 2020 from https://www.acm.org/education/curricula-recommendations

The Joint Task Force on Computing Curricula, IEEE-CS, and ACM. 2001. *Computing Curricula 2001 Computer Science: Final Report.* ACM, New York, NY, USA. Retrieved Oct. 1, 2020 from https://www.acm.org/education/curricula-recommendations

Franklyn Turbak, Mark Sherman, Fred Martin, David Wolber, and Shaileen Crawford Pokress. 2014. Events-First Programming in APP Inventor. *J. Comput. Sci. Coll.* 29, 6 (June 2014), 81–89. Retrieved Oct. 1, 2020 from http://dl.acm.org/doi/10.5555/2602724.2602739

Sharon M. Tuttle. 2001. ¡YO Quiero Java!: Teaching Java as a Second Programming Language. *J. Comput. Sci. Coll.* 17, 2 (Dec. 2001), 34–45. Retrieved Oct. 1, 2020 from http://dl.acm.org/doi/10.5555/775339.775348

Ian Utting. 2006. Problems in the Initial Teaching of Programming Using Java: The Case for Replacing J2SE with J2ME. In *Proc. 11th ITiCSE* (Bologna, IT). ACM, New York, NY, USA, 193–196. https://doi.org/10.1145/1140124.1140176

Haodong Wang. 2014. Engendering Excitement and Interest in Computer Science Courses by Using Emerging Wireless Sensors. *J. Comput. Sci. Coll.* 30, 1 (Oct. 2014), 61–69. Retrieved Oct. 1, 2020 from http://dl.acm.org/doi/10.5555/2667369.2667380

Robert Ward and Martin Smith. 1998. JavaScript as a First Programming Language for Multimedia Students. In *Proc. ITiCSE* (Dublin, IE). ACM, New York, NY, USA, 249–253. https://doi.org/10.1145/282991.283557

David Weintrop, Alexandria K. Hansen, Danielle B. Harlow, and Diana M. Franklin. 2018. Starting from Scratch: Outcomes of Early Computer Science Learning Experiences and Implications for What Comes Next. In *Proc. 2018 ICER* (Espoo, FI). ACM, New York, NY, USA, 142–150. https://doi.org/10.1145/3230977.3230988

Jim Whitehead. 2008. Introduction to Game Design in the Large Classroom. In *Proc. 3rd GDCSE* (Miami, FL, USA). ACM, New York, NY, USA, 61–65. https://doi.org/10.1145/1463673.1463686

Richard Wicentowski and Tia Newhall. 2005. Using Image Processing Projects to Teach CS1 Topics. In *Proc. 36th SIGCSE* (St. Louis, MO, USA). ACM, New York, NY, USA, 287–291. https://doi.org/10.1145/1047344.1047445

Rosalee Wolfe. 1999. New Possibilities in the Introductory Graphics Course for Computer Science Majors. *ACM SIGGRAPH Comput. Graph.* 33, 2 (May 1999), 35–39. https://doi.org/10.1145/326460.326489

Ursula Wolz and Elliot Koffman. 1999. SimpleIO: A Java Package for Novice Interactive and Graphics Programming. In *Proc. 4th ITiCSE* (Cracow, PL). Association for Computing Machinery, New York, NY, USA, 139–142. https://doi.org/10.1145/305786.305896

Pat Woodworth and Wanda Dann. 1999. Integrating Console and Event-Driven Models in CS1. In *Proc. 30th SIGCSE* (New Orleans, LA, USA). ACM, New York, NY, USA, 132–135. https://doi.org/10.1145/299649.299720

Benjamin Xiang-Yu Xie. 2016. *Progression of Computational Thinking Skills Demonstrated by App Inventor Users.* Master's thesis. Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, MA, USA. Retrieved Oct. 1, 2020 from http://hdl.handle.net/1721.1/106395

Ivan Zupic and Tomaž Čater. 2015. Bibliometric Methods in Management and Organization. *Organ. Res. Methods* 18, 3 (July 2015), 429–472. https://doi.org/10.1177/1094428114562629

APPENDICES

The appendices A and B contain a more detailed description of the research methodology as well as the tables and the figure referenced from the main text. These appendices have been published as supplemental material alongside this article.

Event-driven Programming in Programming Education

A Mapping Review

ALEKSI LUKKARINEN, LAURI MALMI, and LASSI HAARANEN

ACM Reference format:
Aleksi Lukkarinen, Lauri Malmi, and Lassi Haaranen. 2021. Event-driven Programming in Programming Education: A Mapping Review. *ACM Trans. Comput. Educ.* 21, 1, Article 1 (March 2021), 31 pages. https://doi.org/10.1145/3423956

APPENDIX A METHODOLOGY: SEARCHING AND RAPID SCREENING

In this appendix, we detail the first two phases of our research method more deeply than we did in Chapter 2 of the base article: We consider the inclusion/exclusion criteria (§ A.1), discuss the *Searching* phase (§ A.2), and conclude by describing the *Rapid Screening* phase (§ A.3). We list all references in the bibliography of the base article.

A.1 Inclusion/Exclusion Criteria

Table A1 below presents the inclusion/exclusion criteria that we followed to constrain the publications that this mapping review covers. The criteria impose restrictions on availability, type, language, publication ranking level (JUFO[1]), and content of the documents to be included to the study. Six of these seven criteria require further elaboration.

• *First Criterion.* Restricting a study to include only sources that are freely available for a specific organization may obviously introduce some biasing, depending on the number and providers of the sources missed because of the restriction. However, as this limitation is not easily circumvented in the scope of this study, it can only be accepted as a part of the limitations of the study.

• *Second Criterion.* Because this study aims to focus on publications that report formal research, the second inclusion/exclusion criterion restricts the study to consider the content from two publication channel types of the formal and strict end of the spectrum of scientific publications in the field of computing education research: journal articles as well as papers in proceedings of conference series. As it is probable that all key results in this discipline of research are published in those two types of publication channels, they will become included to the search results of this study regardless of the exclusion of book chapters and theses, for instance. Moreover, textbooks have a tendency to be written from a specific perspective as well as to simplify and omit information, which is why we do not consider them as suitable sources in this study.

• *Third Criterion.* Restricting a study to sources that are expressed in certain languages may introduce *language bias* [e.g., Coe et al. 2017, p. 174]. Still, as with the first criterion that was

[1] A Finnish classification system that measures the quality of scientific publication channels; see § A.3 for more details.

ID	Topic	Inclusion Criterion	Exclusion Criterion
1	Availability	Documents that are immediately accessible from Aalto University for free.	Documents that have to be requested for or are subject to charge when accessed from Aalto University.
2	Document type	Documents that are journal articles or papers in proceedings of conference series.	Documents that are neither journal articles nor papers in proceedings of conference series. *Examples:* Austing et al. [2002] (technical report), Xie [2016] (thesis), Schliesmann et al. [2004] (patent), Sherwood [1996] (news article), books, presentation slides, and drafts.
3	Language	Documents that are written essentially in English.	Documents that are written essentially in other languages than English. *Examples:* Sungkyung and Sangchul [2018] (only the abstract of a document is written in English) and Papadakis and Orfanakis [2015] (the whole document is written in Greek).
4	Not an introduction	Documents that are not introductions to some primary content.	Documents that only introduce some primary content. *Examples:* Astrachan et al. [2005] (panel discussion), McCauley et al. [2006] (workshop), Fischer [2011] (demonstration), McNally [1998] (poster), and Sherman [2014] (research project of a doctoral consortium participant).
5	About event-driven programming (EDP)	Documents that explicitly use some variant of the term *event-driven programming* in the sense of a computer programming style.	Documents that are not related to EDP in the sense relevant for this study. *Examples:* Documents that are concerned with scheduling social occasions or television broadcasts.
6	About teaching or learning EDP	Documents that are concerned with teaching or learning EDP.	Documents that are not concerned with teaching and learning EDP. *Examples:* Gasiunas et al. [2011] (extension of a programming language), O'Kelly and Gibson [2006] (EDP as an implementation technique of a tool), and Papavlasopoulou et al. [2017] (only background literature is mentioned in relation to events).
7	*Journals:* JUFO level *Conferences:* Continuity (see § A.3)	*Journals:* On at least JUFO level 1 OR other knowledge about a formal peer review. *Conferences:* Has more than a single occurrence.	*Journals:* Articles in journals at JUFO level 0 (assigned either officially or—in case an official classification is missing—by the author for the purposes of this study) AND no knowledge about a formal peer review exist. *Conferences:* Has occurred only once. *Examples:* Chalk [1999] (classified as JUFO 0).

Note: In the seventh criterion, the abbreviation JUFO refers to a Finnish classification to measure the quality of scientific publication channels; see § A.3 for more details.

Table A1. Inclusion/Exclusion Criteria.

discussed above, this limitation cannot be easily evaded (as using commercial translators is out of the question) and thus can only be accepted as it is.

• *Fifth Criterion.* This criterion requires two things. First, the term *event-driven programming*, or its variant, has to be explicitly mentioned in every included publication. This requirement follows from observing that there are two important sets of resources that concern event-driven programming (EDP): (1) Those that explicitly use some variant of the term EDP, and (2) those that do not explicitly mention EDP but implicitly deal with it. The latter set would contain *all* even vaguely suitable content ever produced and thus would be difficult to define exactly (e.g., the meaning of *suitability* depends on context), quite large in size, as well as next to impossible to acquire and deal with in practice. Consequently, this study is limited to the former category.

Second, the included publication has to be related to computer programming, that is, creating instruction sequences for computers to execute. However, the word *programming* has also other meanings, such as scheduling television broadcasts and social occasions. Obviously, we excluded such publications.

• *Sixth Criterion.* In case of a publication that showcases computer software or hardware as a teaching tool (a *"tool paper"*), in that same document, the tool in question has to be mentioned in relation to event-driven programming (EDP).

• *Seventh Criterion.* In parallel with the second criterion (see above), this restriction is implemented to intentionally introduce *database bias* in order to focus this study on publication channels in the higher end (as defined by the criterion of and classified by the expert panels maintaining the JUFO publication rankings; see § A.3) of the quality spectrum.

A.2 Searching for Publications

To find publications to review, we decided to use only automatic searches and omit manual systematical skimming and reading through publication channels. The search expressions (see Table A2 below) were based on the following phrases: *event-driven programming*, *event-based programming*, *event-oriented programming*, *events-first programming*, and *event programming*. Based on few quick preliminary searches, we assumed the number of directly relevant search results to be in the order of several dozens at the most. With that assumption and given the exploratory nature of our study, everything related to event-driven programming in programming education was of interest without the need to further restrict the scope.

An issue with the most of the above phrases is the sheer multiplicity of search results: There is a multitude of publications about topics related to event-driven programming (EDP), but only small portion of that is directly related to teaching or learning EDP in programming education. For instance, the search expression *+"event-driven programming"* in *Google Scholar* (a full-text search including patents and citations) resulted in an approximation of over 7000 search results on the first result page. Similarly, a search using the variant *event-based programming* approximated over 2000 results. Any number of results of this magnitude would have been impossible to deal with.

To limit the scope, we experimented with additional search terms related to teaching and learning programming in general. However, we did not find a justifiable approach to limit the number of full-text search results in respect to the above goal of not restricting the scope to a single narrow subtopic. Consequently, we had to restrict several actual searches to document titles.

With an estimate of potential search expressions, we chose eight search services (see Table A3) for searching publications. The first author (AL) performed the searches (see Table A2) during five consecutive days (September 18th–22nd, 2018). As we expected Google Scholar to index the content of several other relevant search services, such as *Association for Computing Machinery*

ID	Date	Service	Restrictions	Expression	Total	Downloaded		# of Excluded			Included			
						#	%S	T	C	J	#M	#F	$\%S_f$	$\%D_f$
S1	Sep. 18, 2018	Google Scholar	Including patents and citations	+"events-first programming"	28	26	92.9	1	16	0	5	4	14.3	0.6
S2	"	"	"	+"event programming paradigm"	18	17	94.4	13	4	—	—	—	—	—
S3	"	"	"	allintitle: +"event-oriented programming"	5	4	80.0	3	0	0	0	1	20.0	0.2
S4	"	"	"	allintitle: +"event-based programming"	33	23	69.7	21	2	—	—	—	—	—
S5	Sep. 19, 2018	"	"	allintitle: +"event-driven programming"	105	46	43.8	34	7	0	0	5	4.8	0.8
S6	"	"	"	allintitle: +"event programming"	39	14	35.9	12	2	—	—	—	—	—
S7	"	ACM Digital Library	In all fields of metadata, restricted to the *ACM Full-Text Collection*	(+"event-driven programming")	111	91	82.0	55	20	0	3	13	11.7	2.0
S8	"	"	"	(+"event-based programming")	37	27	73.0	19	2	0	0	6	16.2	0.9
S9	"	"	"	(+"event-oriented programming")	2	1	50.0	1	—	—	—	—	—	—
S10	"	"	"	(+"events-first programming")	1	0	—	—	—	—	—	—	—	—
S11	"	"	"	(+"event programming paradigm")	0	—	—	—	—	—	—	—	—	—
S12	"	"	"	(+"event programming")	8	8	100.0	6	2	—	—	—	—	—
S13	"	IEEE Xplore	In conferences, journals & magazines, books, early access articles, and standards	(((((”Document Title”:”event-driven programming”) **OR** ”Document Title”:”event-based programming”) **OR** ”Document Title”:”event-oriented programming”) **OR** ”Document Title”:”event programming”) **OR** ”Document Title”:”events-first programming”)	15	1	6.7	1	—	—	—	—	—	—
S14	"	Elsevier ScienceDirect	In full text of everything except books and book reviews	introductory **AND** (”event-driven programming” **OR** ”event-based programming” **OR** ”event-oriented programming”)	14	14	100.0	3	3	0	3	5	35.7	0.8
S15	Sep. 20, 2018	"	"	(curriculum **OR** curricula **OR** syllabus **OR** syllabi) **AND** (”event-driven programming” **OR** ”event-based programming” **OR** ”event-oriented programming”)	20	9	45.0	2	4	0	1	2	10.0	0.3
S16	"	Springer Nature SpringerLink	In full text of everything, excluding preview-only content	introductory **AND** (”event-driven programming” **OR** ”event-based programming” **OR** ”event-oriented programming”)	23	14	60.9	4	5	0	3	2	8.7	0.3
S17	"	"	"	education **AND** (”event-driven programming” **OR** ”event-based programming” **OR** ”event-oriented programming”)	80	69	86.3	48	17	0	0	4	5.0	0.6
S18	"	Wiley Online Library	In full text of journals	(course **OR** novice **OR** introductory **OR** education **OR** taxonomy **OR** syllabus **OR** curriculum) **AND** (”event-driven programming” **OR** ”event-based programming” **OR** ”event-oriented programming” **OR** ”event-programming paradigm” **OR** ”events-first programming”)	53	46	86.8	33	8	0	3	2	3.8	0.3
S19	"	Taylor & Francis Online	In full text of everything	”event-driven programming” **OR** ”event-based programming” **OR** ”event-oriented programming” **OR** ”event-programming paradigm” **OR** ”events-first programming”	50	39	78.0	20	8	0	5	6	12.0	0.9
S20	Sep. 21, 2018	ACM Digital Library	In the *ACM Full-Text Collection*	content.ftsec:(”event-driven programming” ”event-based programming” ”event-oriented programming” ”event programming paradigm” ”events-first programming”) **AND** acmdlCCS:(education)	214	193	90.2	4	59	3	74	53	24.8	8.2
S21	Sep. 22, 2018	ERIC	—	”event-driven programming” **OR** ”event-based programming” **OR** ”event-oriented programming” **OR** ”event programming paradigm” **OR** ”events-first programming”	2	1	50.0	0	0	0	0	1	50.0	0.2
Totals	5 days	8 services			858	643		280	159	3	97	104		16.2

Notes: $\%S$ = percentage in total results of the search in question; T / C / J = excluded results by title, by content, and by JUFO (see § A.3) level; $\#M$ / $\#F$ = number of marginally/fully included results; $\%S_f$ = percentage of fully included results in total results of the search in question; $\%D_f$ = percentage of fully included results in downloaded results of all searches combined.

Table A2. Automated searches and the numbers of their results (see § A.2).

(Acm) and *Institute of Electrical and Electronics Engineers* (Ieee), we decided to start with it and complement the results from other services.

Table A3. Numbers of search results by search service (see § A.2).

Service	Total	Downloaded		# of Excluded			Included			
		#	%S	T	C	J	#M	#F	$\%S_f$	$\%D_f$
ACM Digital Library https://dl.acm.org/	373	320	85.8	85	83	3	77	72	19.3	11.2
Google Scholar https://scholar.google.com/	228	130	57.0	84	31	0	5	10	4.4	2.0
Elsevier ScienceDirect https://www.sciencedirect.com/	34	23	67.6	5	7	0	4	7	20.6	1.1
Springer Nature SpringerLink https://link.springer.com/	103	83	80.6	52	22	0	3	6	5.8	0.9
Taylor & Francis Online https://www.tandfonline.com/	50	39	78.0	20	8	0	5	6	12.0	0.9
Wiley Online Library https://onlinelibrary.wiley.com/	53	46	86.8	33	8	0	3	2	3.8	0.3
ERIC https://eric.ed.gov/	2	1	50.0	0	0	0	0	1	50.0	0.2
IEEE Xplore https://ieeexplore.ieee.org/	15	1	6.7	1	—	—	—	—	—	—
Totals	858	643		280	159	3	97	104		16.2

Notes: *% S* = percentage in total results returned by the search service in question
T / C / J = excluded results by title, by content, and by Jufo level (see § A.3)
 # M / # F = number of marginally/fully included results
$\% S_f$ = percentage of fully included results in total results returned by the search service in question
$\% D_f$ = percentage of fully included results in downloaded results of all searches services combined

For each search service, we tried different combinations of search expressions and targeted data fields from the more general to the more restricted, while we observed the number of results to limit it to approximately 300 results per search. We recorded a search run when a quick glance at a search result page with an acceptable number of results revealed resources with relevant-looking titles. We skipped over citations as well as commercial books and book chapters, but accepted other resources that were immediately accessible and not discovered yet during these searches.

For storing data about the search runs, the first author used a *Microsoft Excel* workbook as a relational database: One spreadsheet contained general characteristics of each recorded search run,[2] and second one had information of all search results[3] that he accepted as candidates for inclusion. After completing the whole search process, he exploited *Power Query* in Excel to develop queries based on the above data model to derive statistics and charts for the search runs.

[2] Details recorded for each recorded search run were: Identification number, Date, Search service, Restrictions, Search expression, and Number of total results.

[3] Details recorded for each accepted search result were: ID number of the related search run, Hyperlink, and Notes.

With the above procedure, the actual searches produced a large number of results (858 in total; 643 (74.9 %) downloaded and screened) compared to the number of publications that we finally included to the study (104; 12.1 % of the total). As we judged both the total number of search results and the number of included search results to be high considering the scope of this study, we did not want to increase the number of results by searching [e.g., Kitchenham et al. 2016, p. 64] for publications that cite or are referenced by the included publications.

From Table A2, it can be seen that of all the search runs recorded, *S20* returned the largest number of included results (n=53, 51.0 % of the 104 included results) while the number of results returned was still manageable for this study. The most obvious reason for this is the incorporation of the *Acm Computing Classification System*[4] (ccs) into the search expression, which enabled the search engine to perform a full-text search while focusing on the documents whose authors had classified them as education-related. Thus, as a hindsight, it would have been better to start by searching *Acm Digital Library* with this sort of search expression, and continue to complement those results with the other search services. Google Scholar seems to be worthwhile in finding grey literature, but it lacks either the extensive metadata offered, for instance, by many publishers' own search engines or just the ability to restrict the search to some parts of it.

Another matter visible in Table A2 is that at first, search expressions contained only one single phrase at a time *(S1–S12),* whereas the latter searches *(S13–S21)* combined several phrases into one search. This was due to the higher numbers of results returned by Google Scholar, but after noticing the smaller result numbers returned by the next search service *(S7–S12),* it was decided to try combining the phrases together to simplify the search process by grouping search results and thus reducing the amount of search runs that had to be recorded. This practice was continued through the rest of the searches.

A.3 Rapid Screening

The *Rapid Screening* phase consisted of three stages: Preliminary Review, Content Check, and Jufo publication ranking level check (see below). During these stages, we enforced the include/exclude criteria (see above) on the downloaded search results and recorded the decisions about exclusion.

• *Preliminary Review.* During this first stage, we excluded search results that either (1) we knew to be unacceptable in advance on the basis of checking them during the preliminary searches or (2) had a title that clearly looked irrelevant in respect to the goals of this study. If we could not reach a decision based on the title or prior knowledge, we left the result in question to be handled later in the screening process. The inclusion/exclusion criteria (see Table A1 above) were not yet developed into their final forms, but the decisions about excluding search results were roughly based on the same criteria. A few examples of titles representing excluded results are listed in Figure 2 in the base article. On this stage, 280 search results were excluded, which consists of 32.6 % of the total 858 results found. This left 363 results (42.3 % of the total) for the next stage.

• *Content Check.* On this stage, we quickly checked the publications for inappropriate content types, such as technical reports, abstracts, theses, patents, and unknown languages. As a result, we excluded 44 search results (5.1 % of the total of 858); a breakdown of reasons for exclusion is presented with the related result counts in Figure 2 in the base article. In total, 319 results (37.2 %) were left for the third and last *Rapid Screening* stage. Not all inappropriate publications were detected during this screening stage, however, and a few more publications were excluded for their content types later during the *Detailed Screening* phase (see Figure 3).

[4] See: https://www.acm.org/about-acm/class, accessed October 1, 2020.

• *JUFO Level Check.* Our purpose was to focus this study on publications from channels that represent the higher end of the quality spectrum of those that—in the broadest sense—publish scientific information. To this end, we applied this third stage of *Rapid Screening* to restrict inclusion based on a Finnish classification system developed for assessing the quality of scientific publication channels. This system is known by the abbreviation JUFO that stands for *Julkaisufoorumi*[5] (*Publication Forum* in English).

The JUFO classification has a single dimension that consists of three hierarchical levels: "Basic," "Leading," and "Highest," numbered from one to three, respectively. Each level has criteria that a publication channel has to meet to be classified on that level; channels that do not meet the Basic level criteria are assigned level number zero. If a publication channel and the related publisher have been classified on the same level, then the channel is not listed separately, and the classification given to the publisher applies. The organization maintaining the classification system operates under *The Federation of Finnish Learned Societies*[6] (*Tieteellisten seurojen valtuuskunta* in Finnish). The actual classifications are given and maintained by expert panels that consist in total of approximately two hundred scholars, who either are Finnish or are working in Finland. The classification database can be queried via Julkaisufoorumi website, and the individual classifications might be revised in time.

In practice, on this stage we compiled the JUFO levels of the search results that passed the two previous stages and ensured that all journals[7] are (1) at least of JUFO level one (see criterion 7 in Table A1) or (2) otherwise known to be peer-reviewed. Although this check applies only to journals, JUFO levels were compiled for all search results in the interest of analysis and completeness. All JUFO levels were recorded during Nov. 13, 2019, and for all but the few most recent years it would be impossible to use the publication-time JUFO levels. Most of the publication channels that were represented in the search results had an official JUFO class. For the rest, authors' personal judgement was used to unofficially classify them for the purposes of this study. Based on the JUFO levels, we excluded three publications, and 316 search results (36.8 %) were left for the next phase.

[5] See: http://www.julkaisufoorumi.fi/en/publication-forum, accessed October 1, 2020.
[6] See: https://www.tsv.fi/en, accessed October 1, 2020.
[7] In this study, the *Journal of Computing Sciences in Colleges* is interpreted to be a series of conference proceedings.

Event-driven Programming in Programming Education

A Mapping Review

ALEKSI LUKKARINEN, LAURI MALMI, and LASSI HAARANEN

ACM Reference format:
Aleksi Lukkarinen, Lauri Malmi, and Lassi Haaranen. 2021. Event-driven Programming in Programming Education: A Mapping Review. *ACM Trans. Comput. Educ.* 21, 1, Article 1 (March 2021), 31 pages.
https://doi.org/10.1145/3423956

APPENDIX B RESULT-RELATED TABLES AND FIGURES

To offer more in-depth knowledge about the results of this review, this appendix provides the following tables and figures:

Title (⛰ and ⛰ denote historical/semi-historical publications; see § 3.3. 💎 denotes one of the 25 most-cited publications; see § 3.1.2 and Table B3.)	First Author	Year	General Pedagogical Offerings	ISCED Levels	Course Context; *Programming Theme*	Programming Languages and Tools	
						General-Purpose	Educational
⛰ ¡YO quiero Java!: Teaching Java as a Second Programming Language	Tuttle S. M.	2001	👁	6	CS2; *Web*	Java	—
.NET Gadgeteer: A New Platform for K–12 Computer Science Education	Hodges S. *8*	2013	🔧	3	K–12; *Embedded*	C#, VB	.NET Gadgeteer
A "CS 1.5" Introduction to Web Programming	Stepp M. *3*	2009	👁	6	Web programming	JavaScript, PHP, SQL	—
⛰ A CS1 to CS2 Bridge Class Using 2D Game Programming	Leutenegger S. T.	2006	👁	6	CS1/CS2; *Games*	ActionScript	—
A Different Approach of Teaching Introductory Visual Basic Course	Jiang K. *3*	2004	👁	6	CS1; *GUIs*	VB, VBA, VBScript	—
💎 A Game-Based Approach to the Teaching of Object-Oriented Programming Languages	Rodríguez Corral J. M. *5*	2014	🔧	6	CS2	C#	Sifteo Cubes
A Graphics Package for the First Day and Beyond	Goldwasser M. H. *2*	2009	🔧	3, 6	CS1; *Graphics*	Python	cs1graphics
⛰ A Hypertext Module for Teaching User Interface Design	Barrett M. L.	1993	👁 🔧	6	Graphics/GUIs	—	—
💎 ⛰ A Library to Support a Graphics-Based Object-First Approach to CS 1	① Bruce K. B. *3*	2001	🔧	6	CS1; *Graphics*	Java	objectdraw
A New Approach to Computer Science in the Liberal Arts	Burns B.	2005	👁	6	CS1	—	DrRacket
A Placement Test for Computer Science: Design, Implementation, and Analysis	Nugent G. *4*	2006	🔧	6	CS1	—	—
⛰ A Required Mechanical Engineering Course in Microprocessors	McNeill S. R. *2*	1995	👁	6	Mech. eng., μ-proc.	*Machine lang. in hex*	—
⛰ A Tangible Interface for Controlling Robotic Toys	Patten J. *3*	2000	🔧	—	K–12; *Robots*	—	Lego Mindstorms
A Top-Down Approach to Teaching Introductory Computer Graphics	Sung K. *2*	2004	👁	6	Graphics/GUIs	C++	—
⛰ Adapting Computer Graphics Curricula to Changes in Graphics	Hitchner L. E. *2*	2000	👁	6	Graphics/GUIs	Java	—
Adapting Higher-Order List Operators for Blocks Programming	④ Kim S. *2*	2015	🔧 ➡	6	Mobile programming	—	App Inventor
Adding Breadth to CS1 and CS2 Courses through Visual and Interactive Programming Projects	Jimenez-Peris R. *3*	1999	👁	6	CS1/CS2; *Graphics, Games*	—	—
AdMoVeo: A Robotic Platform for Teaching Creative Programming to Designers	Alers S. *2*	2009	🔧	6	Creative programming for designers; *Robots*	Java (Processing)	AdMoVeo
An Environment for Learning Interactive Programming	Tang T. *3*	2014	🔧	6	CS1	Python	Skulpt, CodeSkulptor, SimpleGUI
Alice: A 3-D Tool for Introductory Programming Concepts	Cooper S. *3*	2000	👁 🔧	3, 6	CS0/CS1	Python	Alice
💎 Assessment of Computer Science Learning in a Scratch-Based Outreach Program	② Franklin D. *15*	2013	👁 ➡	2	Outreach	—	Scratch
Beautiful JavaScript: How to Guide Students to Create Good and Elegant Code	Passier H. *3*	2014	👁	—	—	JavaScript	—
BlockyTalky: New Programmable Tools to Enable Students' Learning Networks	Kelly A. *4*	2018	👁 🔧 ➡	2	K–12; *Embedded, Music, Network*	—	BlockyTalky
Chief Chefs of Z to Alloy: Using a Kitchen Example to Teach Alloy with Z	Tarkan S. *2*	2009	👁 🔧 ➡	6, 7, 8	Formal methods	Alloy, Z	—

Notes: Pedagogical offerings: 👁 = approaches, 🔧 = tools, ➡ = outcomes. Language acronyms: BASIC = Beginner's All-Purpose Symbolic Instruction Code; PHP = PHP: Hypertext Preprocessor; SQL = Structured Query Language; VB(A) = Visual Basic (for Applications). The circled numbers left from the authors refer to the author-based publication streams; see § 3.2 and Figure B4. Numbers on the right side of authors refer to the total number of authors, when more than one exist.

Continues on the next page...

Table B1. Substantial data collected from the publications fully included into this study (see § 2.2).

Title (▲ and △ denote historical/semi-historical publications; see § 3.3. 💎 denotes one of the 25 most-cited publications; see § 3.1.2 and Table B3.)	First Author	Year	General Pedagogical Offerings	ISCED Levels	Course Context; *Programming Theme*	Programming Languages and Tools	
						General-Purpose	Educational
Collateral Learning of Mobile Computing: An Experience Report	Kumar A. N.	2018	👁	6, 7	Artificial intelligence, Prog. lang.; *Mobile*	*(Java, C++)*	—
Combining Big Data and Thick Data Analyses for Understanding Youth Learning Trajectories in a Summer Coding Camp	③ Fields D. A. *4*	2016	➧	1, 2	Outreach	—	Scratch
💎 Controversy on How to Teach CS 1: A Discussion on the SIGCSE-Members Mailing List	① Bruce K. B.	2004	👁	6	CS1	—	—
▲ Creating Windows Applications Using Borland's OWL Classes	Szuecs L.	1996	🔧	6	Graphics/GUIs	C++	—
💎 Debugging: A Review of the Literature from an Educational Perspective	McCauley R. *7*	2008	*lit. review*	—	—	—	—
Designing a Relational Social Robot Toolkit for Preschool Children to Explore Computational Concepts	Gordon M. *4*	2015	👁 🔧	0	Preschool computing	—	Social Robot Toolkit (incl. DragonBot)
💎 Developing Assessments to Determine Mastery of Programming Fundamentals	Luxton-Reilly A. *10*	2017	👁	6	CS1; *GUIs*	—	—
Developing Principles of GUI Programming Using Views	Bishop J. *2*	2004	🔧	—	CS curriculum	C#	Views
DoodlePad: Next-Gen Event-Driven Programming for CS1	Russo M. F.	2017	🔧	6	CS1; *Graphics*	Java	DoodlePad
△ Easy, Realistic GUIs with Java in CS1	Lambert K. *2*	2000	🔧	3, 6	CS1/CS2; *GUIs*	Java	BreezyGUI
Ecological Design-Based Research for Computer Science Education: Affordances and Effectivities for Elementary School Students	② Harlow D. B. *5*	2018	👁 🔧 ➧	1	K–6	—	LaPlaya
EcoSim: A Language and Experience Teaching Parallel Programming in Elementary School	Gregg C. *4*	2012	👁 🔧	1	K–6; *Parallelism*	—	EcoSim
💎 Educating for Mobile Computing: Addressing the New Challenges	Burd B. *6*	2012	👁 🔧	—	CS curriculum; *Mobile*	JavaScript	AppInventor, TouchDevelop
Effects of Oral Metaphors and Allegories on Programming Problem Solving	Hidalgo-Céspedes J. *4*	2018	👁 ➧	6	CS1	Java	Goldbach Calculator
Engaging Students in Programming Fundamentals Using Alice 2.0	Mullins P. M. *2*	2008	👁	6	CS1	Java	Alice
Engendering Excitement and Interest in Computer Science Courses by Using Emerging Wireless Sensors	Wang H.	2014	👁	6, 7	Oper. syst., Security; *Wireless sensors*	NesC	—
Enhancing Robot Programming with Visual Feedback and Augmented Reality	Magnenat S. *4*	2015	🔧 ➧	3	K–12; *Augmented reality, Robots*	—	Thymio II, Aseba/VPL
▲ Event-Driven Programming Facilitates Learning Standard Programming Concepts	① Bruce K. B. *2*	2004	👁	3, 6	CS1	Java	objectdraw
💎 Event-Driven Programming Is Simple Enough for CS1	① Bruce K. B. *3*	2001	👁	6	CS1; *GUIs*	—	
Events Not Equal to GUIs	Hansen S. *2*	2004	👁	6, 7	Event-driven progr.	Java	—

Notes: Pedagogical offerings: 👁 = approaches, 🔧 = tools, ➧ = outcomes. Language acronyms: BASIC = Beginner's All-Purpose Symbolic Instruction Code; PHP = PHP: Hypertext Preprocessor; SQL = Structured Query Language; VB(A) = Visual Basic (for Applications). The circled numbers left from the authors refer to the author-based publication streams; see § 3.2 and Figure B4. Numbers on the right side of authors refer to the total number of authors, when more than one exist.

Continues on the next page...

Table B1 *(continued)*. Substantial data collected from the publications fully included into this study (see § 2.2).

Title (◭ and ◬ denote historical/semi-historical publications; see § 3.3. 💎 denotes one of the 25 most-cited publications; see § 3.1.2 and Table B3.)	First Author	Year	General Pedagogical Offerings	ISCED Levels	Course Context; *Programming Theme*	Programming Languages and Tools	
						General-Purpose	Educational
Events-First Programming in App Inventor	④ Turbak F. *5*	2014	🔧	–	CS0; *Mobile*	–	App Inventor
◭ Experiences with CCB-Directed Projects in the Classroom	Purtilo J. *2*	1994	👁	6	SE project course	–	–
Experiencing Aspects of Games Programming in an Introductory Computer Graphics Class	Sung K. *3*	2007	👁	6	Graphics/GUIs; *Games*	–	–
💎 ◭ Exploring the Difficulties of Learning Object-Oriented Techniques	Sheetz S. D. *5*	1997	🡆	6, 7	Advanced progr.	Smalltalk	–
◭ Frameworks in CS1: A Different Way of Introducing Event-Driven Programming	Christensen H. B. *2*	2002	👁 🔧	6	CS1; *GUIs*	Java	Presenter Framework
Frogs to Think with: Improving Students' Computational Thinking and Understanding of Evolution in a Code-First Learning Environment	Guo Y. *6*	2016	👁 🔧 🡆	2	K–9 science education	–	FrogPond
From Alice to Python. Introducing Text-Based Programming in Middle Schools	Tabet N. *4*	2016	👁 🡆	2	K–12	Java, Python	Alice
From Android Games to Coding in C—An Approach to Motivate Novice Engineering Students to Learn Programming: A Case Study	Dolgopolovas V. *3*	2018	👁 🡆	6	CS1; *Mobile*	C	App Inventor
How to Creatively Communicate Microsoft.NET Technologies in the IT Curriculum	Chaytor L. *2*	2003	👁	5	CS curriculum	VB, JavaScript	–
◭ How to Teach Graphics Using X (and Live to Tell about It)	Pavlidis T.	1996	🔧	6	Graphics/GUIs	C	–
Incorporating Tangible Computing Devices into CS1	Goadrich M.	2014	👁	6	CS1; *Hardware, Embedded, Mobile*	Python	App Inventor
Initialization in Scratch: Seeking Knowledge Transfer	② Franklin D. *6*	2016	🡆	1	K–6	C, Java	Scratch
◭ Instrumentation: A Multi-Science Integrated Sequence	Tesser H. *3*	2000	👁	6	CS1; *Processing data from instruments*	VB, VBA	–
Integrating Console and Event-Driven Models in CS1	Woodworth P. *2*	1999	👁	6	CS1	C++	–
◭ Introducing Concurrency in CS 1	① Bruce K. B. *3*	2010	👁	6	CS1; *Concurrency*	Java	objectdraw
◭ Introducing High School Students to Event Driven Programming	Lang R. R. *2*	1999	👁	3	Outreach	ToolBook, Open Script	
Introducing OO Design and Programming with Special Emphasis on Concrete Examples	Angster E. *3*	1999	👁	6	CS1	–	–
Introduction to Game Design in the Large Classroom	Whitehead J.	2008	👁	6	Game design/progr.	–	GameMaker
Introductory Programming in a Web Context	de Raadt M.	2010	👁	6	CS1; *Web*	JavaScript	–
◭ JavaScript as a First Programming Language for Multimedia Students	Ward R. *2*	1998	👁	6	CS1; *Multimedia*	JavaScript	–
💎 Learning through Game Modding	El-Nasr M. S. *2*	2006	👁	3, 6	Game design/progr.; *Games*	C#, UnrealScript	–
Let's Modify the Objects-First Approach into Design-Patterns-First	Pecinovský R. *3*	2006	👁	2, 3, 6	K–12, CS1, Complem. educ.; *Design patterns*	Java	BlueJ

Notes: Pedagogical offerings: 👁 = approaches, 🔧 = tools, 🡆 = outcomes. Language acronyms: BASIC = Beginner's All-Purpose Symbolic Instruction Code; PHP = PHP: Hypertext Preprocessor; SQL = Structured Query Language; VB(A) = Visual Basic (for Applications). The circled numbers left from the authors refer to the author-based publication streams; see § 3.2 and Figure B4. Numbers on the right side of authors refer to the total number of authors, when more than one exist.

Continues on the next page...

Table B1 *(continued)*. Substantial data collected from the publications fully included into this study (see § 2.2).

Title (▲ and ▲ denote historical/semi-historical publications; see § 3.3. 💎 denotes one of the 25 most-cited publications; see § 3.1.2 and Table B3.)	First Author	Year	General Pedagogical Offerings	ISCED Levels	Course Context; *Programming Theme*	Programming Languages and Tools	
						General-Purpose	Educational
MakeCode and CODAL: Intuitive and Efficient Embedded Systems Programming for Education	Devine J. *6*	2019	🔧	—	K–12; *Embedded*	Static Type Script, C++	—
Mediating Programming through Chat for the OLPC	Dimond J. P.	2009	🔧 ➲	1, 2	Outreach	*A chat with a small bespoke programming language*	
💎 Modeling the Learning Progressions of Computational Thinking of Primary Grade Students	Seiter L. *2*	2013	👁 ➲	1	K–6	—	Scratch
💎 My Program Is Correct But It Doesn't Run: A Preliminary Investigation of Novice Programmers' Problems	Garner S. *3*	2005	➲	6	CS1	Java	—
▲ New Possibilities in the Introductory Graphics Course for Computer Science Majors	Rosalee Wolfe	1999	👁	6	Graphics/GUIs	C++	—
💎 No Sensor Left Behind: Enriching Computing Education with Mobile Devices	Dabney M. H. *3*	2013	👁	3	Outreach; *Mobile*	Java	App Inventor
Object-Oriented Programming at Upper Secondary School for Advanced Students	Lubomir Salanci	2006	👁	3	K–12	Object Pascal	—
Parallel Programming with Pictures is a Snap!	Feng A. *3*	2017	👁 🔧	2	Outreach; *Parallelism*	—	Snap!
Playing Beowulf: Bridging Computational Thinking, Arts and Literature through Game-Making	de Paula B. H. *4*	2018	➲	2	Outreach; *Games*	—	MissionMaker
Problems in the Initial Teaching of Programming Using Java: The Case for Replacing J2SE with J2ME	Utting I.	2006	🔧	—	CS1/CS2; *GUIs*	Java	—
💎 Programming in the Wild: Trends in Youth Computational Participation in the Online Scratch Community	③ Fields D. A. *3*	2014	➲	1, 2, 3	K–12	—	Scratch
Reflections on Teaching App Inventor for Non-Beginner Programmers: Issues, Challenges and Opportunities	④ Soares A.	2014	👁	6, 7	Mobile programming	—	App Inventor
💎 Skill Progression Demonstrated by Users in the Scratch Animation Environment	Scaffidi C. *2*	2012	➲	1, 2, 3	K–12	—	Scratch
Squint: Barely Visible Library Support for CS1	① Murtagh T. P.	2007	👁 🔧	6	CS1; *GUIs, Networks*	Java	Squint
Starting from Scratch: Outcomes of Early Computer Science Learning Experiences and Implications for What Comes Next	② Weintrop D. *4*	2018	👁 ➲	1	K–12		LaPlaya
Students' Experiences from the Use of MIT App Inventor in Classroom	Perdikuri K.	2014	👁 ➲	3	K–12	—	App Inventor
subTextile: Reduced Event-Oriented Programming System for Sensate Actuated Materials	Sadi S. *2*	2007	🔧	—	Complementary education	—	subTextile
Teaching CS1 with Web Applications and Test-Driven Development	Schaub S.	2009	👁	6	CS1; *Web*	C#	—
💎 ▲ Teaching Design Patterns through Computer Game Development	Gestwicki P. *2*	2008	👁 🔧	6	Design patterns; *Games*	Java	EEClone
Teaching Embedded Software Concepts Using Android	Muppala J. K.	2011	👁	6	Embedded progr.	Java	—

Notes: Pedagogical offerings: 👁 = approaches, 🔧 = tools, ➲ = outcomes. Language acronyms: BASIC = Beginner's All-Purpose Symbolic Instruction Code; PHP = PHP: Hypertext Preprocessor; SQL = Structured Query Language; VB(A) = Visual Basic (for Applications). The circled numbers left from the authors refer to the author-based publication streams; see § 3.2 and Figure B4. Numbers on the right side of authors refer to the total number of authors, when more than one exist.

Continues on the next page...

Table B1 *(continued)*. Substantial data collected from the publications fully included into this study (see § 2.2).

Title (▲ and △ denote historical/semi-historical publications; see § 3.3. 💎 denotes one of the 25 most-cited publications; see § 3.1.2 and Table B3.)	First Author	Year	General Pedagogical Offerings	ISCED Levels	Course Context; *Programming Theme*	Programming Languages and Tools	
						General-Purpose	Educational
Teaching Non-Beginner Programmers with App Inventor: Survey Results and Implications	④ Soares A. *2*	2015	👁 🡺	6	Mobile programming	—	App Inventor
💎 Teaching Objects-First in Introductory Computer Science	Cooper S. *3*	2003	👁	6	CS1	Java, C++	Alice
Teaching Software Design Engineering Across the K–12 Curriculum: Using Visual Thinking and Computational Thinking	Fronza I. *3*	2016	👁	1, 2, 3	K–12 curriculum	—	—
The Assessment of Mobile Computational Thinking	④ Sherman M. *2*	2015	👁	6	Mobile programming	—	App Inventor
The CCL-Parallax Programmable Badge: Learning with Low-Cost, Communicative Wearable Computers	Brady C. *5*	2015	👁 🔧	6	CS curriculum	BASIC, C, Spin, Assembler	CCL-Parallax
💎 The Curse of Monkey Island: Holding the Attention of Students Weaned on Computer Games	Ladd B. C.	2006	👁	6	CS1/CS2; *Games*	C++	—
The Design of Mobile Apps: What and How to Teach?	Stuurman S. *3*	2014	👁	6, 7	CS curriculum; *Mobile*	—	—
The Impact of Different Teaching Approaches and Languages on Student Learning of Introductory Programming Concepts	Kunkle W. M. *2*	2016	🡺	6	CS1	Java, VB, C++	BlueJ
💎 The Programmers' Collective: Fostering Participatory Culture by Making Music Videos in a High School Scratch Coding Workshop	③ Fields D. A. *3*	2015	👁 🡺	3	K–12; *Media*	—	Scratch
💎 Using a Discourse-Intensive Pedagogy and Android's App Inventor for Introducing Computational Concepts to Middle School Students	Grover S. *2*	2013	👁	2	K–12; *Mobile*	—	App Inventor
💎 Using Alice 2.0 as a First Language	Mullins P. *3*	2009	👁 🡺	6	CS1/CS2	Java, C++	Alice
💎 ▲ Using Graphics to Support the Teaching of Fundamental Object-Oriented Principles in CS1	Alphonce C. *2*	2003	👁	6	CS1; *Graphics*	Java	NGP
💎 Using Image Processing Projects to Teach CS1 Topics	Wicentowski R. *2*	2005	👁	6	CS1; *Media*	Java, C	—
💎 Using Mobile Phone Programming to Teach Java and Advanced Programming to Computer Scientists	Riley D.	2012	👁	6	Advanced progr.; *Mobile*	Java	—
Using Open-Source Robocode as a Java Programming Assignment	Liu P. L.	2008	👁 🡺	6	Advanced progr.; *Games*	Java	Robocode
💎 Using Second Life in Programming's Communities of Practice	Esteves M. *5*	2008	👁	6	CS1/CS2; *Virtual reality*	Second Life, Linden Scripting Language	—
△ Why Structural Recursion Should Be Taught before Arrays in CS 1	① Bruce K. B. *3*	2005	👁	6	CS1	Java	objectdraw
▲ XDP: A Simple Library for Teaching a Distributed Programming Module	Arnow D. M.	1995	🔧	6	Operating systems	C	XDP
Youth Computational Participation in the Wild: Understanding Experience and Equity in Participating and Programming in the Online Scratch Community	③ Fields D. A. *3*	2017	🡺	—	K–12	—	Scratch

Notes: Pedagogical offerings: 👁 = approaches, 🔧 = tools, 🡺 = outcomes. Language acronyms: BASIC = Beginner's All-Purpose Symbolic Instruction Code; PHP = PHP: Hypertext Preprocessor; SQL = Structured Query Language; VB(A) = Visual Basic (for Applications). The circled numbers left from the authors refer to the author-based publication streams; see § 3.2 and Figure B4. Numbers on the right side of authors refer to the total number of authors, when more than one exist.

Table B1 *(continued)*. Substantial data collected from the publications fully included into this study (see § 2.2).

Publication Channel	Abbreviations	Type	Publisher	# of Results	JUFO Level	Source of JUFO Level
SIGCSE Technical Symposium https://sigcse.org/sigcse/events/symposia	SIGCSE	Conference	ACM	26	1	Publisher
Innovation and Technology in Computer Science Education (*formerly* Integrating Technology into Computer Science Education; *includes* ITiCSE Working Group Reports) http://iticse.acm.org/	ITiCSE, ITiCSE-WGR	Conference	ACM	11	1	Channel
Journal of Computing Sciences in Colleges (*formerly* Journal of Computing in Small Colleges) http://www.ccsc.org/	J Comput Sci Coll (JCSC)	Journal (proceedings of conferences)	CCSC	11	0	Authors' judgement
Information Technology Education (*formerly* Information Technology Curriculum) https://www.sigite.org/	SIGITE, CITC4, CITC5	Conference	ACM	5	1	Publisher
ACM Transactions in Computing Education (*formerly* Journal on Educational Resources in Computing) https://toce.acm.org/ (http://jeric.acm.org/)	ACM Trans Comput Educ (TOCE), J Educ Resour Comput (JERIC)	Journal	ACM	3	2	Channel

Summary 5 publication channels (3 journals and 2 conferences) of JUFO levels 0–2, consisting of 56 (53.3 %) of the 105 included publications.

Notes: Several publication channels have been renamed. *J Comput Sci Coll* is included on the basis of interpreting it as a series of conference proceedings. The abbreviation JUFO refers to a Finnish classification to measure the quality of scientific publication channels; see § A.3. All JUFO levels were collected during November 13, 2019.

Table B2. Five most-used publication channels in the 105 fully included publications according to the first versions of the publications (see § 3.1.1).

Citations	Title (⛰ and ⛰ denote historical/semi-historical publications; see § 3.3)	Authors	First Publication (channels with JUFO level ≠ 1 are accentuated)				
			Year	Channel	JUFO	Publisher	Pages
297	Alice: A 3-D Tool for Introductory Programming Concepts	Cooper S., Dann W., Pausch R.	2000	*J Comput Sci Coll*	*0* A	CCSC	10
248 ✦	Teaching Objects-First In Introductory Computer Science	Cooper S., Dann W., Pausch R.	2003	SIGCSE	1 P	ACM	5
123	Learning through Game Modding	El-Nasr M. S., Smith B. K.	2006	ACM Comput Entertain	1 C	"	20
82	Modeling the Learning Progressions of Computational Thinking of Primary Grade Students	Seiter L., Foreman B.	2013	ICER	1 P	"	8
73	Debugging: A Review of the Literature from an Educational Perspective	McCauley R. *et al.* *(7 authors)*	2008	**Comput Sci Educ**	**2** C	Taylor & Francis	26
56 ✧	My Program Is Correct But It Doesn't Run: A Preliminary Investigation of Novice Programmers' Problems	Garner S., Haden P., Robins A. V.	2005	ACE	1 P	ACS	8
51	Assessment of Computer Science Learning in a Scratch-Based Outreach Program	② Franklin D. *et al.* *(15 authors)*	2013	SIGCSE	1 P	ACM	6
47 ✦	Controversy on How to Teach CS 1: A Discussion on the SIGCSE-Members Mailing List	① Bruce K. B.	2004	ITiCSE-WGR	1 P	"	7
44	A Game-Based Approach to the Teaching of Object-Oriented Programming Languages	Rodríguez Corral J. M. *et al.* *(5 authors)*	2014	**Comput Educ**	**3** C	Elsevier	10
40	Programming in the Wild: Trends in Youth Computational Participation in the Online Scratch Community	③ Fields D. A., Giang M., Kafai Y. B.	2014	WiPSCE	1 P	ACM	10
39	⛰ Teaching Design Patterns through Computer Game Development	Gestwicki P., Sun F.	2008	**J Educ Res Comput**	**2** C	"	21
35	Skill Progression Demonstrated by Users in the Scratch Animation Environment	Scaffidi C., Chambers C.	2012	Int J Hum Comput Interact	1 C	Taylor & Francis	16
32 ✧	Using a Discourse-Intensive Pedagogy and Android's App Inventor for Introducing Computational Concepts to Middle School Students	Grover S., Pea R.	2013	SIGCSE	1 P	ACM	6
32	Using Alice 2.0 as a First Language	Mullins P., Whitfield D., Conlon M.	2009	*J Comput Sci Coll*	*0* A	CCSC	8
31	⛰ Exploring the Difficulties of Learning Object-Oriented Techniques	Sheetz S. D. *et al.* *(5 authors)*	1997	**J Manag Inf Syst**	**3** C	Taylor & Francis	29
30	⛰ A Library to Support a Graphics-Based Object-First Approach to CS 1	① Bruce K. B., Danyluk A. P., Murtagh T. P.	2001	SIGCSE	1 P	ACM	5
30	Developing Assessments to Determine Mastery of Programming Fundamentals	Luxton-Reilly A. *et al.* *(10 authors)*	2017	ITiCSE-WGR	1 P	"	23
27	The Programmers' Collective: Fostering Participatory Culture by Making Music Videos in a High School Scratch Coding Workshop	③ Fields D. A., Vasudevan V., Kafai Y. B.	2015	Interact Learn Environ	1 C	Taylor & Francis	21
25	Using Mobile Phone Programming to Teach Java and Advanced Programming to Computer Scientists	Riley D.	2012	SIGCSE	1 P	ACM	6
24	The Assessment of Mobile Computational Thinking	Sherman, Mark and Martin, Fred	2015	*J Comput Sci Coll*	*0* A	CCSC	7
23	Educating for Mobile Computing: Addressing the New Challenges	Burd B. *et al.* *(6 authors)*	2012	ITiCSE-WGR	1 P	ACM	13
22	⛰ Using Graphics to Support the Teaching of Fundamental Object-Oriented Principles in CS1	Alphonce C., Ventura P.	2003	OOPSLA	1 P	"	6
22	Using Image Processing Projects to Teach CS1 Topics	Wicentowski R., Newhall T.	2005	SIGCSE	1 P	"	5
21	No Sensor Left Behind: Enriching Computing Education with Mobile Devices	Dabney M. H., Dean B. C., Rogers T.	2013	"	1 P	"	6
21	The Impact of Different Teaching Approaches and Languages on Student Learning of Introductory Programming Concepts	Kunkle W. M., Allen R. B.	2016	**ACM Trans Comput Educ**	**2** P	"	26
Summary	25 (23.8 %) of the 105 included results; 3 of these are historical and 1 is semi-historical	86 unique authors; 1–15 per result	1997–2017	15 channels; 7 SIGCSEs	0–3	5 publishers	5–29

Notes: ✦/✧ = citations do/might come from multiple publications. All citation counts were collected (see § 2.2.2) from Scopus during Sept. 3, 2020. Some citation records from Scopus did not contain a citation count and are not taken into account. If they were counted as single citations, however, their effect would be about 1–5 citations per result, which makes the large-scale effect practically negligible. The numbers next to the authors refer to the author-based publication streams; see § 3.2 and Figure B4. The abbreviation JUFO refers to a Finnish classification to measure the quality of scientific publication channels; see § A.3. All JUFO levels were collected during Nov. 13, 2019. The letters in the JUFO column indicate the source of the level as follows: *C* = publication channel, *P* = publisher, *S* = publication series, and *A* = author's judgement.

Table B3. 25 most-cited publications in the 105 fully included publications (see § 3.1.2).

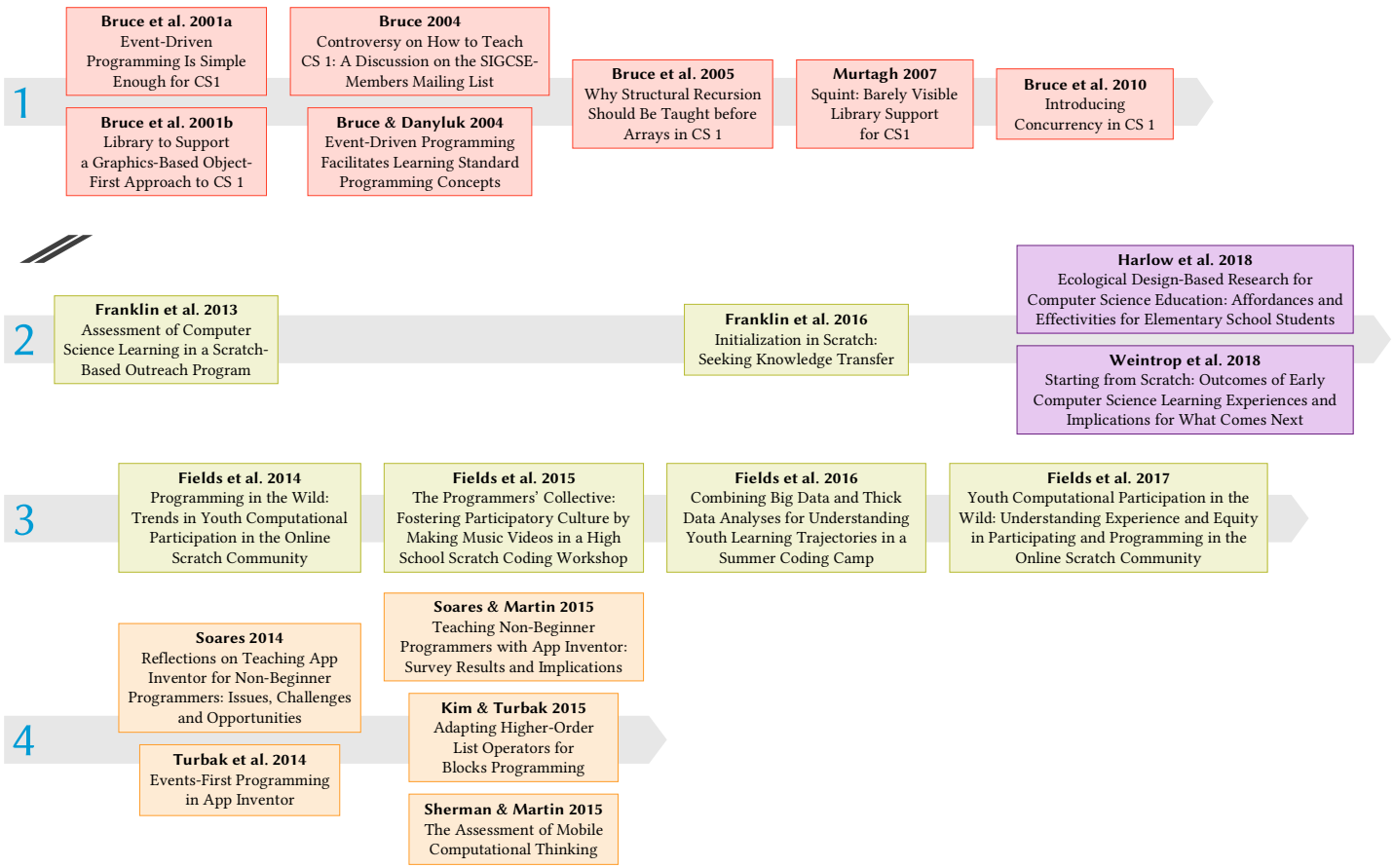

Fig. B4. Four author-based publication streams having at least four publications (see § 3.2). The first stream *(red)* is related mostly to ObjectDraw and Squint. The second and third streams are concerned with Scratch *(green)*, except Harlow et al. [2018] and Weintrop et al. [2018] *(purple)*, which deal with LaPlaya. Finally, the fourth stream *(orange)* is linked to App Inventor. Please notice that the timeline continues from the first stream to the other three streams.