

Sequencing and Paradigm-Transition Evidence in Object-Oriented Programming: A Systematic Review

Fredrick Michael Ogore

fogore@daystar.ac.ke

Daystar University Nairobi School of Science Engineering and Health <https://orcid.org/0000-0001-7099-6761>

Systematic Review

Keywords: object-oriented programming, introductory programming, objects-first, curriculum sequencing, paradigm transition, systematic review

Posted Date: October 27th, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-11492416/v1>

License: © ⓘ This work is licensed under a Creative Commons Attribution 4.0 International License. [Read Full License](#)

Additional Declarations: The authors declare no competing interests.

Abstract

Whether object-oriented (OO) concepts should be taught before or after procedural programming has been debated for more than two decades. This systematic review, reported according to PRISMA 2020, synthesises empirical studies that compared objects-first with objects-later sequencing, together with studies of how learners move from procedural to OO thinking. Forty documented search strings returned 376 hits; 117 unique records were screened and 16 studies (17 reports, 1997–2021) were included. Quality was appraised with the Mixed Methods Appraisal Tool, and findings were combined through narrative synthesis. Of five direct comparisons, three favoured objects-first and two found no difference in achievement; none favoured objects-later. Every positive result combined sequencing with a visual environment or a small sample, so the effects of sequence and tool cannot be separated. Procedural experience shaped OO learning mainly through misleading analogies and simplified models of method dispatch, while difficulties with classes and objects, references, associations and design persisted under every sequence. No study evaluated an explicit bridge from C structures to classes for learners with prior structured-programming experience. Sequence alone appears to be a weak lever; designs that connect procedural knowledge explicitly to OO constructs warrant rigorous, multi-site evaluation.

1. Introduction

Object-oriented programming (OOP) has been the dominant paradigm in university programming curricula for more than two decades, and it remains central to industrial practice. Yet OOP is widely reported to be difficult to teach and to learn. Recent secondary studies rank “difficulty in understanding, teaching and implementing object-orientation”, “difficulties related to understanding classes”, and “difficulty in understanding object-oriented relationships” as the most serious problems in the field (Gutiérrez et al., 2022), and identify the complexity and abstraction of the material, lecture time, and students’ background as recurring obstacles (Efan et al., 2023).

A long-standing and practical question within this literature concerns *when* OO concepts should be introduced. Under an **objects-first** (or objects-early) approach, learners meet classes and objects at or near the start of their first programming course, often supported by pedagogical environments such as BlueJ or Alice. Under an **objects-later** approach, learners first acquire variables, control structures, functions, and (in C-family languages) structures, and only then move to classes, typically around the middle of a semester or in a subsequent course. The question became the subject of an intense debate within the computing education community in 2004, which was subsequently staged as a formal debate (Astrachan et al., 2005) and analysed by an ITiCSE working group. That analysis concluded that the discussion had relied mainly on personal experience rather than on research evidence, and that the literature at the time neither showed that objects-first made learning harder nor that it made learning simpler (Lister et al., 2006). Influential textbooks continue to take opposing positions, from objects-first designs built around BlueJ to explicitly “back to basics” procedural-first designs (Reges, 2006).

The question is not purely historical. Many institutions, particularly in contexts where C is the first language, teach a structured-programming course followed by a separate OOP course in C++ or Java. In such curricula the issue is no longer whether learners should see objects before loops, but how a course should be sequenced for learners who *already* hold a well-practised procedural mental model. Cognitive accounts suggest that such prior knowledge can either support new learning, by providing schemas to which new concepts can be attached, or hinder it through proactive interference (Underwood, 1957; Sweller, 1988; White & Sivitanides, 2005). Which of these effects dominates, and under what instructional conditions, is an empirical question.

Existing secondary studies of OOP education are broad in scope, covering problems, tools, games, and teaching models (Efan et al., 2023; Gutiérrez et al., 2022), or cover introductory programming as a whole (Luxton-Reilly et al., 2018). To our knowledge, no systematic review has focused specifically on the combined evidence on OO *sequencing* and on the *transition* from procedural to OO thinking. The present review addresses this gap. It asks four research questions:

- **RQ1.** What is the empirical evidence on the effects of objects-first compared with objects-later sequencing on learners' achievement, retention, and attitudes?
- **RQ2.** What does the evidence show about how learners' prior procedural knowledge affects learning OOP?
- **RQ3.** Which OO conceptions remain difficult for learners irrespective of sequencing, and how might these relate to sequencing choices?
- **RQ4.** What are the methodological characteristics and gaps of this body of evidence?

The review contributes (i) a transparent, reproducible synthesis of sixteen empirical studies spanning secondary, undergraduate, graduate, and in-service teacher populations; (ii) a critical analysis of confounds, especially the co-variation of sequencing with pedagogical tools; and (iii) a research agenda, including design features for future trials of sequencing approaches aimed at learners with prior structured-programming experience.

2. Theoretical and Conceptual Background

2.1 Defining objects-first and objects-later

The labels “objects-first” and “objects-later” are used inconsistently. Some courses described as objects-first begin with object *use* (calling methods on pre-built objects) and defer class *definition*; others begin with class design and modelling; still others are objects-first chiefly because they use a visual microworld in which every entity is an object (Xinogalos et al., 2006). Similarly, “objects-later” may mean a procedural first half of a single course, as in the classroom comparison by Ehlert and Schulte (2009), or an entire procedural course followed by a separate OO course. A further variant, *outside-in* teaching, starts from complete applications and works inward to their internal structure (Janke et al., 2015). For this review, we use **objects-first** to denote any sequence in which classes and objects are introduced, used, and practised before or alongside basic control structures, and **objects-later** for any sequence in which a substantial block of procedural instruction precedes them.

2.2 Cognitive perspectives on paradigm transition

Three theoretical lenses are used in this review to interpret findings. First, *cognitive load theory* holds that working memory is severely limited, and that instructional designs which introduce many interacting new elements at once impose extraneous load that impedes schema construction (Sweller, 1988). A mid-semester introduction of classes, access specifiers, constructors, and references in close succession is a plausible example. Second, *interference theory* distinguishes proactive interference, in which earlier learning disrupts later learning, from retroactive interference, in which later learning disrupts earlier learning (Underwood, 1957). White and Sivitanides (2005) explicitly attribute the difficulty of moving from procedural to OO programming to proactive interference rather than to intrinsic cognitive differences between the paradigms. Third, research on *mental models and notional machines* emphasises that learners need an explicit model of program execution; where none is taught, they construct their own, often from everyday analogies or from a previously learned paradigm (Sorva, 2013).

2.3 Prior reviews

Robins et al. (2003) reviewed the broad literature on learning and teaching programming and noted that novices' difficulties lie less in syntax than in planning and in combining constructs. Luxton-Reilly et al. (2018) systematically mapped the introductory-programming literature and identified curriculum and paradigm choice as a long-running but weakly evidenced topic. Within OOP specifically, Gutiérrez et al. (2022) conducted a tertiary review and ranked problems and solutions using multi-criteria decision methods; notably, among the highest-ranked solutions was introducing OOP "at an early point in the curriculum" while emphasising basic programming concepts. Efan et al. (2023) reviewed 60 articles published between 2017 and 2021 and found that most recommendations concerned learning media and learning models. None of these reviews isolated the empirical evidence on sequencing or on the procedural-to-OO transition, which is the focus here.

3. Methodology

3.1 Protocol and reporting

The review followed the PRISMA 2020 statement (Page et al., 2021) and the guidelines for systematic reviews in software engineering and computing by Kitchenham and Charters (2007). Because quantitative outcomes were too heterogeneous for meta-analysis, synthesis followed the Synthesis Without Meta-analysis (SWiM) reporting guideline (Campbell et al., 2020). The protocol was not registered in advance; this is acknowledged as a limitation (Section 5.3). The eligibility criteria, search strings, and screening decisions are reported in full (Table 1, Appendix A, and the supplementary screening log) so that the review can be reproduced and updated.

3.2 Eligibility criteria

Eligibility criteria are summarised in Table 1. Studies were included if they reported primary empirical data (quantitative, qualitative, or mixed) on human learners and addressed either (a) a comparison of OO sequencing approaches, or (b) the transition from procedural to OO programming, the influence of procedural prior knowledge on OO learning, or learners' conceptions and difficulties with core OO concepts (class, object, reference, inheritance, polymorphism, association, OO design). Position papers, panels, textbooks, experience reports without systematically collected outcome data, tool descriptions without learner outcomes, and secondary studies were excluded (secondary studies were used as background).

Table 1
Eligibility criteria.

Criterion	Inclusion	Exclusion
Population	Learners of programming in secondary, tertiary, or professional/teacher-education settings	Pre-school and primary learners; expert professionals studied for comprehension only
Focus	OO sequencing (objects-first/early vs. objects-later/procedural-first; outside-in); procedural-to-OO transition; conceptions and difficulties with core OO concepts	Language comparisons not involving paradigm or sequence; general programming difficulties not specific to OO
Study type	Primary empirical study (quantitative, qualitative, or mixed methods)	Position papers, panels, textbooks, experience reports without systematic data, tool descriptions without learner outcomes, secondary studies
Outcomes	Achievement, comprehension, misconceptions, design quality, retention/progression, attitudes or affect	None reported, or outcomes unrelated to OO learning
Publication	Peer-reviewed journal article, conference or workshop paper, or doctoral thesis; English or German	Non-scholarly web pages; reports not retrievable for full assessment
Period	Any date up to 1 October 2026	—

3.3 Information sources and search strategy

Searches were executed on 1 October 2026 through a general-purpose academic web search engine that indexes, among others, the ACM Digital Library, IEEE Xplore, ScienceDirect, SpringerLink, Taylor & Francis, SAGE, ERIC, Semantic Scholar, ResearchGate, arXiv, and institutional repositories. Forty search strings were used (Appendix A). They combined paradigm terms (“objects-first”, “objects early”, “objects-later”, “procedural”, “imperative-first”, “outside-in”), learning terms (“misconceptions”, “difficulties”, “conceptions”, “comprehension”, “transition”), concept terms (“class”, “object”, “inheritance”, “polymorphism”, “this”), and context terms (“CS1”, “introductory”, “BlueJ”, “Alice”, “C++”, “Java”, “Python”). Some strings were designed to locate studies already known to be relevant or cited in retrieved reports (backward citation tracking), for example studies referenced in the working-group report by Lister et al. (2006). Publisher landing pages, institutional repositories, ERIC records, and author-hosted PDFs were then used to retrieve reports and verify bibliographic details.

3.4 Selection process

All hits were first de-duplicated at the level of the scholarly work (for example, an ACM record, a ResearchGate copy, and a Semantic Scholar entry of the same paper were counted once), and non-scholarly pages (retail listings, document-sharing sites, forums, encyclopaedia entries, slide decks, and course pages) were removed. Remaining records were screened on title and available abstract against Table 1. Reports passing this stage were sought in full; when full text was not accessible, the publisher abstract and indexing record were used if they contained sufficient design and outcome information, otherwise the report was classified as “not retrieved”. Screening was conducted by a single reviewer with AI-assisted support (see the declaration at the end of the paper); all inclusion decisions were checked against the retrieved text.

3.5 Data collection

For each included study we extracted: citation; country and setting; population and sample size; programming language and environment; study design; sequencing condition or focus; outcome measures; and principal findings,

including statistics where reported. Where a study was reported in more than one publication (Ehlert & Schulte, 2007, 2009), data were extracted from all reports and attributed to one study.

3.6 Quality appraisal

Methodological quality was appraised with the Mixed Methods Appraisal Tool (MMAT), version 2018 (Hong et al., 2018), which provides five design-specific criteria for qualitative, quantitative randomised, quantitative non-randomised, quantitative descriptive, and mixed-methods studies. Each criterion was rated “yes”, “no”, or “can’t tell”. Consistent with MMAT guidance, no overall numeric score was used to exclude studies; instead, the number of criteria met and the principal concern are reported for each study (Table 3) and are taken into account in the synthesis.

3.7 Synthesis methods

Because designs, populations, and outcome measures were heterogeneous and most studies did not report the statistics needed to compute standardised effect sizes, meta-analysis was not appropriate. For RQ1, studies were grouped by comparison type and summarised by *direction of effect* (favours objects-first, no clear difference, favours objects-later), a vote-counting approach recommended by SWiM when effect sizes cannot be pooled (Campbell et al., 2020). For RQ2 and RQ3, findings were synthesised thematically: reported difficulties and explanations were coded, grouped into clusters, and compared across populations and sequencing contexts. For RQ4, study characteristics and MMAT ratings were tabulated and gaps identified.

4. Results

4.1 Study selection

The selection process is summarised in Fig. 1. The 40 search strings returned 376 hits. After removing 259 duplicates and non-scholarly pages, 117 unique scholarly records were screened on title and abstract, of which 62 were excluded, most commonly because they were off-topic (for example, language comparisons not involving paradigm or sequence), non-empirical (textbooks, panels, position and experience papers), or concerned with tools without learner outcomes. Of the 55 reports sought in full, 29 could not be retrieved in a form sufficient for assessment (for example, because of access restrictions on publisher and repository pages); these are listed in the supplementary screening log. Of the 26 reports assessed for eligibility, 9 were excluded: five reported no primary learner data (a curriculum-material analysis, a position paper, an analysis of practitioner discourse, a commentary on CS1 content approaches, and a debate panel), two were tool evaluations without sequencing or transition outcomes, and two were secondary studies. Sixteen studies, reported in seventeen publications, were included.

4.2 Characteristics of included studies

Table 2 summarises the included studies. They were published between 1997 and 2021 and were conducted in the United States ($n = 6$), Israel ($n = 3$), Germany ($n = 2$), Canada ($n = 1$), Greece ($n = 1$), Norway ($n = 1$), and the United Kingdom ($n = 1$); the setting of one study was not stated in the accessible report. Populations included secondary-school students ($n = 2$ studies, plus one study combining secondary and college students), undergraduates ($n = 10$), graduate students ($n = 1$), in-service teachers ($n = 1$), and professional programmers ($n = 1$). Java was the dominant language ($n = 10$); C++ was used in two studies and Alice in two. Five studies directly compared sequencing approaches (RQ1); six examined the procedural-to-OO transition or compared comprehension across paradigms (RQ2); and five characterised OO conceptions and difficulties within OO-focused courses (RQ3), several of which were objects-first.

Table 2
Characteristics of included studies ($k = 16$).

#	Study	Setting; population (n)	Language / tool	Design	Focus	Principal findings
1	Ehlert & Schulte (2007, 2009)	Germany; two parallel grade-11 classes, vocational upper-secondary school (≈ 40)	Java	Quasi-experiment, one school year; pre-, post- and 8-week follow-up tests	OOP-first vs. OOP-later, same content, different order	No overall difference in learning outcomes; post-test differences favoured OOP-first only for arrays/strings and associations; no difference at follow-up; $\approx 94\%$ retention in both groups
2	Uysal (2012)	USA; graduate students in an introductory OOP course (20)	Java / BlueJ	Quasi-experiment, 13 weeks; pre/post achievement test	Objects-first (visual) vs. objects-late (text-based start)	Higher achievement for objects-first (Mann–Whitney $z = -2.29, p = .022$)
3	Janke, Brune & Wagner (2015)	Germany; three university courses (exam participants: 449, 12, 144)	Java	Comparative case study; surveys, exams, interviews	Outside-in vs. traditional	No significant difference in learning outcomes; outside-in students reported less exam anxiety, quicker grasp of the OO paradigm, and clearer course structure
4	Moskal, Lurie & Cooper (2004)	USA; first-year CS majors “at risk” (treatment 25; at-risk control 30; non-at-risk control 52)	Alice	Quasi-experiment; grades, retention, attitudes, focus groups	Objects-first preparatory course with Alice vs. none	Treatment outperformed at-risk controls in CS1 grade (3.03 vs. 1.94) and retention to CS2 (88% vs. 47%); improved attitudes
5	Sykes (2007)	Canada; CS1 students	Alice	Comparative study; performance scores, surveys, observation	Alice-based (objects-first visual) CS1 vs. comparison groups	Alice group outperformed comparison groups ($F(1, 93) = 30.32, p < .001$; $F(1, 81) = 4.18, p = .044$)
6	Nelson, Irwin & Monarchi (1997)	USA; programmers from novice to experienced	Various	Qualitative	Learning OO after procedural experience	Five learner categories followed different paths, met different obstacles, and

#	Study	Setting; population (n)	Language / tool	Design	Focus	Principal findings
						used different strategies
7	White & Sivitanides (2005)	USA; information-systems majors (36)	Java	Correlational	Cognitive predictors of OO vs. procedural performance	Propositional logic predicted OO performance; hemispheric style did not; transition difficulty attributed to proactive interference
8	Lieberman, Beeri & Ben-David Kolikant (2011)	Israel; in-service CS teachers with procedural backgrounds (22)	Java	Qualitative/mixed; observation, questionnaire	Inheritance and polymorphism after procedural experience	≈ 60 errors and misconceptions in four clusters: simplistic models, analogies (everyday and procedural), hierarchy modelling, unresolved basic OO concepts
9	Wiedenbeck, Ramalingam, Sarasamma & Corritore (1999)	USA; novices in OO or procedural courses	OO and procedural languages as taught	Two experiments	Comprehension of OO vs. procedural programs	Short programs: no overall difference, OO learners better on function questions; long programs: procedural learners better on all question types
10	Ramalingam & Wiedenbeck (1997)	USA; novices in an introductory C++ course (75)	C++	Within-subject experiment	Comprehension of imperative vs. OO style	OO programs yielded stronger domain-level (data flow, function) knowledge but higher overall error rates (23.6% vs. 13.1%)
11	Kaasbøll et al. (2004)	Norway; university introductory Java students	Java	Qualitative observation (≈ 28 h)	OO learning after procedural basics	Most students coped reasonably well with OO concepts; one displayed a purely procedural model; students modelled domains implicitly rather

#	Study	Setting; population (n)	Language / tool	Design	Focus	Principal findings
						than through explicit analysis
12	Ragonis & Ben-Ari (2005)	Israel; high-school students, two years	Java / BlueJ	Qualitative, longitudinal	Conceptions in an objects-first course	58 conceptions and difficulties in four categories (class vs. object; instantiation and constructors; simple vs. composed classes; program flow); fundamentals eventually grasped
13	Sanders & Thomas (2007)	Setting not stated; students in an objects-first course (16 students; 71 programs)	Java	Qualitative analysis of programs	OO concepts and misconceptions	Class/instance conflation, hierarchy and abstraction problems, modelling failures, problems linking objects; few constructor or encapsulation problems
14	Thomasson, Ratcliffe & Thomas (2006)	UK; first-year students (180; 67; 90)	UML-style design on paper	Descriptive, three phases	Novice OO design errors	Non-referenced classes in 97% of phase-1 designs; misplaced attributes common; references to non-existent classes in 28%
15	Xinogalos (2015)	Greece; undergraduates, long-term study	Java	Mixed methods	Conceptions of object and class	Roughly half of students comprehended the modelling and static/dynamic aspects of objects and classes
16	Shmallo & Ragonis (2021)	Israel; high-school (86) and college engineering students (77)	Java	Diagnostic questionnaire	Understanding of the this reference	Nine misconceptions; both groups struggled to conceptualise this as the current object

4.3 Methodological quality

MMAT appraisals are reported in Table 3. Quality was moderate overall. The three qualitative studies of OO conceptions (Lieberman et al., 2011; Ragonis & Ben-Ari, 2005; Sanders & Thomas, 2007) and the descriptive studies with larger samples (Shmallo & Ragonis, 2021; Thomasson et al., 2006) met most criteria. The weakest evidence came from the sequencing comparisons, which were all non-randomised. Their main concerns were small or non-equivalent groups (Uysal, 2012: $n = 20$; Ehlert & Schulte, 2009: two intact classes), confounding of sequence with other differences such as tools or institutions (Janke et al., 2015; Moskal et al., 2004; Sykes, 2007; Uysal, 2012), and incomplete reporting of measurement properties (for example, Moskal et al., 2004, noted reliability problems with their content test). No study was randomised at the student level.

Table 3
Quality appraisal with the Mixed Methods Appraisal Tool (MMAT, 2018).

Study	MMAT category	Criteria met (of 5)	Principal concern
Ehlert & Schulte (2007, 2009)	Quantitative non-randomised	3	Two intact classes (≈ 40 students); teacher/class effects cannot be separated
Uysal (2012)	Quantitative non-randomised	2	$n = 20$, all male; visual vs. text interface confounded with sequence
Janke et al. (2015)	Mixed methods	3	Different institutions and courses as comparison groups; unequal sizes
Moskal et al. (2004)	Mixed methods	3	Non-equivalent groups; small treatment group; content-test reliability issues
Sykes (2007)	Mixed methods	2	Comparison groups and integration of methods insufficiently described in accessible report
Nelson et al. (1997)	Qualitative	3	Professional population; limited reporting of data-collection procedures in accessible report
White & Sivitanides (2005)	Quantitative descriptive	3	$n = 36$; interference explanation inferred, not tested
Lieberman et al. (2011)	Qualitative	4	Specific population (in-service teachers); single course
Wiedenbeck et al. (1999)	Quantitative non-randomised	4	Intact course groups; program-length manipulation limited to selected programs
Ramalingam & Wiedenbeck (1997)	Quantitative non-randomised	4	Short, fixed exposure times; single institution
Kaasbøll et al. (2004)	Qualitative	3	Observations limited to groups whose instructors consented; researcher dual role
Ragonis & Ben-Ari (2005)	Qualitative	4	Single school context; transferability limited
Sanders & Thomas (2007)	Qualitative	4	Small sample (16 students)
Thomasson et al. (2006)	Quantitative descriptive	4	Single institution; paper-based design tasks
Xinogalos (2015)	Mixed methods	3	Details of sampling and integration not fully reported in accessible record
Shmallo & Ragonis (2021)	Quantitative descriptive	4	Convenience samples; diagnostic instrument specific to this
Note. MMAT = Mixed Methods Appraisal Tool (Hong et al., 2018). Ratings are based on the accessible report for each study.			

4.4 RQ1: Effects of objects-first versus objects-later sequencing

Five studies compared sequences directly (Table 4). Using the direction-of-effect approach, three favoured objects-first on their primary achievement or progression outcome (Moskal et al., 2004; Sykes, 2007; Uysal, 2012), two found no overall difference (Ehlert & Schulte, 2009; Janke et al., 2015), and none favoured objects-later.

Table 4
Direction of effect in direct sequencing comparisons.

Study	Comparison	Primary outcome	Direction	Sequence isolated from tool?
Ehlert & Schulte (2009)	OOP-first vs. OOP-later (same content)	Achievement (post, follow-up)	No overall difference (two topics favoured OOP-first at post-test only)	Yes
Janke et al. (2015)	Outside-in vs. traditional	Exam achievement; affect	No difference in achievement; affective outcomes favoured outside-in	Partly (different institutions)
Uysal (2012)	Objects-first vs. objects-late	Achievement	Favours objects-first	No (visual vs. text start)
Moskal et al. (2004)	Alice objects-first preparation vs. none	CS1 grade; retention	Favours objects-first	No (Alice and extra course)
Sykes (2007)	Alice-based CS1 vs. comparison	Performance scores	Favours objects-first	No (Alice environment)

Two observations qualify this apparently favourable pattern. First, the only comparison that held content, duration, and tooling constant and varied only the order of topics (Ehlert & Schulte, 2007, 2009) found that both sequences produced comparable learning, with long-term retention of about 94% in each group. Its topic-level differences at post-test (arrays/strings and associations, favouring OOP-first) disappeared at follow-up. Second, all three positive findings confound sequence with a visual or microworld environment, and two of them with additional instruction (Moskal et al., 2004) or a small, homogeneous sample (Uysal, 2012). The evidence is therefore consistent with the view that objects-first sequencing is *not harmful* and may support progression and attitudes when combined with supportive environments, but it does not establish that the sequence itself causes higher achievement. This conclusion is in line with the results of studies we could not retrieve in full but which are described in secondary sources as finding no significant difference between procedural and OO CS1 cohorts (Vilner et al., 2007; see also the studies summarised by Lister et al., 2006).

Affective and progression outcomes deserve emphasis. Janke et al. (2015) found no achievement difference but reported reduced exam anxiety, a quicker grasp of the OO paradigm, and a clearer perceived course structure under outside-in teaching. Moskal et al. (2004) reported large differences in retention to CS2 among at-risk students (88% vs. 47%). For curriculum designers, such outcomes may be as important as achievement scores.

4.5 RQ2: Prior procedural knowledge and the transition to OOP

Six studies addressed how procedural experience bears on OO learning, and their findings converge on three points.

Prior knowledge is not uniformly a barrier. Kaasbøll et al. (2004) observed university students who had learned procedural basics first and found that most coped “reasonably well” with OO concepts, with only one displaying a purely input–processing–output model. Nelson et al. (1997) found that programmers moving to OO followed several distinct learning paths, meeting different obstacles and adopting different strategies, which suggests that a single transition model does not fit all learners.

Where procedural knowledge interferes, it does so through analogies and simplified execution models. In the most detailed transition study, Liberman et al. (2011) identified around sixty errors and misconceptions about inheritance and polymorphism among 22 experienced teachers with Pascal, C, or BASIC backgrounds. Two of their four clusters

link directly to prior knowledge: analogies to everyday life and to procedural programming that misrepresent inheritance mechanics, and simplistic alternative models, such as believing that the declared type of a variable rather than the run-time type of the object determines which method is invoked. White and Sivitanides (2005) found that formal reasoning (propositional logic) predicted OO performance and argued, without directly testing it, that difficulties in the transition reflect proactive interference from earlier procedural learning.

The paradigms foreground different kinds of knowledge. Two comprehension experiments showed that novices form different mental representations of OO and procedural code. OO programs supported domain-level knowledge (function and data flow) but produced higher overall error rates, while imperative programs supported program-level knowledge of operations and control flow (Ramalingam & Wiedenbeck, 1997). For longer programs, learners taught procedurally outperformed OO learners on all question types (Wiedenbeck et al., 1999), consistent with a steeper initial learning curve for OO and the cognitive demands of following control flow distributed across classes.

4.6 RQ3: Persistent OO conceptual difficulties

Five studies (and the transition studies above) documented OO difficulties in detail. Across populations, languages, and sequences, the difficulties grouped into five recurring clusters (Table 5).

Table 5
Clusters of persistent OO difficulties across included studies.

Cluster	Typical manifestations	Supporting studies
Class–object distinction	Conflating class and instance; treating a class as a set of objects; confusion about instantiation and constructors	Ragonis & Ben-Ari (2005); Sanders & Thomas (2007); Xinogalos (2015)
References and the current object	Misconceptions about object references and this as the current object, particularly in constructors and parameter passing	Shmallo & Ragonis (2021); Ragonis & Ben-Ari (2005)
Dynamic dispatch and hierarchies	Static-type view of method selection; misuse or avoidance of inheritance; confusing inheritance with composition	Liberman et al. (2011); Sanders & Thomas (2007)
Relationships and associations	Classes created but not connected (non-referenced classes); difficulty linking objects; associations rated among the hardest topics	Thomasson et al. (2006); Sanders & Thomas (2007); Ehlert & Schulte (2007)
Modelling and program flow	Copying example structures without domain understanding; implicit rather than explicit modelling; difficulty following flow across objects	Sanders & Thomas (2007); Kaasbøll et al. (2004); Ragonis & Ben-Ari (2005); Wiedenbeck et al. (1999)

Three features of this evidence are relevant to sequencing. First, several of these difficulties were documented in **objects-first** courses (Ragonis & Ben-Ari, 2005; Sanders & Thomas, 2007), so early exposure alone does not prevent them. Second, Sanders and Thomas (2007) found relatively few problems with constructors and encapsulation, concepts that objects-first courses practise from the start, whereas difficulties concentrated on relationships, hierarchies, and modelling. Third, associations emerged as a hard topic in both a design study (Thomasson et al., 2006) and the sequencing comparison (Ehlert & Schulte, 2007), suggesting that the relational aspects of OO design require sustained, explicit instruction whichever sequence is used.

4.7 RQ4: Methodological characteristics and gaps

The evidence base is small, heterogeneous, and dated: 11 of the 16 studies were published before 2010, and only one study published after 2015 met the inclusion criteria. Direct sequencing comparisons were non-randomised, typically

single-site, and often confounded sequence with tooling. Outcome measures were mostly locally developed achievement tests; no study used a validated, language-independent OO concept inventory, and only one included a delayed follow-up (Ehlert & Schulte, 2007). Affective and progression outcomes were measured in only two comparative studies.

Most importantly for curricula in which C precedes an OO course, *no included study evaluated a sequencing approach designed for learners who already know structured programming*, such as an explicit bridge from C structures to classes followed by teaching control flow inside methods. The transition studies examined learners after a procedural phase but did not manipulate how the transition was taught. Populations from Africa, South and South-East Asia, and Latin America were absent from the included studies.

5. Discussion

5.1 Principal findings

This review set out to consolidate the empirical evidence on OO sequencing and on the transition from procedural to OO thinking. Four principal findings emerge. First, direct comparisons do not show that objects-later sequencing produces better learning; three of five favoured objects-first and two found no difference. Second, the positive findings for objects-first are confounded with visual environments and other supports, and the single design that isolated sequence found equivalent outcomes. Sequence alone therefore appears to be a weak lever on achievement, although it may matter for affect and progression. Third, procedural prior knowledge shapes OO learning in specific, identifiable ways: through analogies and simplified execution models that mislead learners about dynamic dispatch and hierarchies, and through a preference for program-level rather than domain-level representations. Fourth, a core set of OO difficulties (class–object distinctions, references, dynamic dispatch, associations, and modelling) persists across sequences and populations.

5.2 Interpretation through cognitive theory

The pattern is consistent with the theoretical lenses introduced in Section 2.2. If difficulties arise mainly from the *intrinsic* complexity of OO concepts and from *extraneous* load created by how they are presented (Sweller, 1988), then changing only the order of topics should have limited effect, which is what Ehlert and Schulte (2009) found. Conversely, interventions that reduce extraneous load, such as visual environments that make objects and their state visible, would be expected to help, which may explain the positive effects associated with Alice and BlueJ (Moskal et al., 2004; Sykes, 2007; Uysal, 2012).

The transition findings suggest a more nuanced account of prior knowledge than a simple “interference” view. Procedural knowledge did not prevent most learners from acquiring OO concepts (Kaasbøll et al., 2004), but it supplied analogies and default models that, when left unaddressed, produced systematic misconceptions (Liberman et al., 2011). This is consistent with notional-machine research, which argues that learners construct their own execution models when no explicit model is taught (Sorva, 2013). The implication is that prior procedural knowledge is a resource whose effect depends on whether instruction connects it explicitly and accurately to OO constructs. A struct-based schema, for example, is a reasonable anchor for the idea that an object groups data, but it can mislead learners about method dispatch and object identity unless those differences are taught explicitly.

5.3 Limitations

Several limitations must be acknowledged. Most importantly, the search was conducted through a general academic web search engine rather than directly in bibliographic databases such as Scopus, Web of Science, ERIC, the ACM

Digital Library, and IEEE Xplore; search-engine results are not fully reproducible, and relevant studies may have been missed. Second, 29 potentially relevant reports could not be retrieved in full, partly because of access restrictions on publisher and repository pages; some, such as the CS1 case study by Vilner et al. (2007), are likely to be relevant, and their exclusion may have biased the synthesis. Third, for some included studies data were extracted from abstracts, publisher records, or accessible summaries rather than full text, which limited extraction and appraisal (noted in Table 3). Fourth, screening, extraction, and appraisal were performed by a single reviewer, supported by AI tools, without independent double coding; inter-rater reliability could therefore not be estimated. Fifth, the protocol was not pre-registered. Sixth, the small number and heterogeneity of studies precluded meta-analysis and limited the strength of conclusions; vote counting ignores effect magnitude and precision. Finally, the author has developed a class-first teaching approach for learners with prior C experience, which constitutes a potential intellectual conflict of interest (see Declarations).

6. Implications for Research and Practice

6.1 Implications for practice

Bearing in mind the moderate quality and small size of the evidence base, the findings support several practical recommendations.

1. **Do not expect sequence alone to raise achievement.** Moving classes earlier, without changing how they are taught, is unlikely to transform outcomes. Investment is better directed at how concepts are presented and practised.
2. **Make objects and their state visible.** The strongest positive results co-occurred with environments that visualise objects. Lightweight equivalents, such as object diagrams, memory-tracing exercises, and interactive object inspection, are available in any language.
3. **Bridge from procedural knowledge explicitly, and mark the limits of the analogy.** For learners who know C, presenting a class as a structure that holds its own functions and protects its own data is a plausible bridge. The transition evidence indicates, however, that the bridge must be complemented by explicit teaching of what does *not* carry over, particularly object identity, references, and run-time dispatch.
4. **Teach an explicit model of dynamic dispatch.** Because static-type misconceptions are common even among experienced programmers (Lieberman et al., 2011), polymorphism should be taught with an explicit execution model, for example a table of function pointers per class, which also connects to C learners' knowledge of function pointers.
5. **Give relationships and modelling sustained attention.** Associations, linking objects, and domain modelling were difficult across studies and sequences; cumulative projects in which classes must collaborate may provide the repeated, contextualised practice these topics require.
6. **Measure affect and progression, not only achievement.** Anxiety, perceived structure, and retention differed between approaches even when achievement did not.

6.2 Implications for research

The gaps identified in RQ4 suggest a focused agenda. First, there is a need for **randomised or well-matched multi-site trials** that vary sequence while holding tools, content, and instructors constant, or that cross sequence with tool support in a factorial design so that their effects can be separated. Second, studies should evaluate **sequencing designs for learners with prior structured-programming experience**, for example a struct-to-class bridge with control flow taught inside methods, against a conventional mid-semester introduction, in curricula where C precedes C + + or

Java. Third, outcome sets should include **validated concept inventories**, measures of design quality, delayed follow-up, and affective and progression outcomes. Fourth, research should include **under-represented regions**, where C-first curricula are common and where class sizes and resources may alter what works. Finally, reports should provide the descriptive statistics needed for future meta-analysis.

7. Conclusion

After more than two decades of debate, the empirical evidence does not support the claim that objects-later sequencing yields better learning than objects-first, nor does it show convincingly that the order of topics, on its own, improves achievement. Positive effects of objects-first teaching co-occur with visual environments and supportive course designs, and the only study that varied order alone found equivalent outcomes. What the evidence shows more clearly is *how* learners struggle: with class–object distinctions, references, dynamic dispatch, relationships, and modelling, and, for learners with procedural experience, with analogies and execution models carried over from that experience. Curriculum designers should therefore focus less on whether objects come first and more on how OO concepts are connected to learners’ prior knowledge and made visible. For the many institutions where C precedes an OO course, explicitly bridged, class-first designs are a promising but as yet untested option, and they warrant rigorous, multi-site evaluation.

Declarations

Ethics Approval

Not applicable.

Informed Consent

Not applicable.

Competing Interests

The author has developed and teaches a class-first approach to object-oriented programming for learners with prior C experience, which is related to the topic of this review. The author declares no financial competing interests.

Author Contributions

This is a single-author manuscript. Conceptualization, Methodology, Investigation, Data Curation, Formal Analysis, Writing – Original Draft, and Writing – Review & Editing, F.M.O.

Data Availability Statement

All data supporting the findings of this study are available within the article and its supplementary material. The search strings are listed in Appendix A, and the full screening log, with decisions and exclusion reasons for every record, is provided as a supplementary file.

References

1. Astrachan O, Bruce K, Koffman E, Kölling M, Reges S (2005) Resolved: Objects early has failed. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education* ACM
2. Campbell M, McKenzie JE, Sowden A, Katikireddi SV, Brennan SE, Ellis S, Hartmann-Boyce J, Ryan R, Shepperd S, Thomas J, Welch V, Thomson H (2020) Synthesis without meta-analysis (SWiM) in systematic reviews: Reporting guideline. *BMJ* 368:l6890. <https://doi.org/10.1136/bmj.l6890>
3. Efan K, Jama J, Mulya R (2023) A systematic literature review of teaching and learning on object-oriented programming course. *Int J Inform Educ Technol* 13(2):302–312. <https://doi.org/10.18178/ijiet.2023.13.2.1808>
4. Ehlert A, Schulte C (2007) Unterschiede im Lernerfolg von Schülerinnen und Schülern in Abhängigkeit von der zeitlichen Reihenfolge der Themen (OOP-First und OOP-Later). *Informatik und Schule – INFOS 2007*. Gesellschaft für Informatik, pp 121–132
5. Ehlert A, Schulte C (2009) Empirical comparison of objects-first and objects-later. In *Proceedings of the Fifth International Workshop on Computing Education Research (ICER '09)*. ACM. <https://doi.org/10.1145/1584322.1584326>
6. Gutiérrez LE, Guerrero CA, López-Ospina HA (2022) Ranking of problems and solutions in the teaching and learning of object-oriented programming. *Educ Inform Technol* 27:7205–7239. <https://doi.org/10.1007/s10639-022-10929-5>
7. Hong QN, Fàbregues S, Bartlett G, Boardman F, Cargo M, Dagenais P, Gagnon M-P, Griffiths F, Nicolau B, O’Cathain A, Rousseau M-C, Vedel I, Pluye P (2018) The Mixed Methods Appraisal Tool (MMAT) version 2018 for information professionals and researchers. *Educ Inform* 34(4):285–291. <https://doi.org/10.3233/EFI-180221>
8. Janke E, Brune P, Wagner S (2015) Does outside-in teaching improve the learning of object-oriented programming? In *Proceedings of the 37th IEEE/ACM International Conference on Software Engineering (ICSE 2015), Software Engineering Education and Training track*. IEEE. Preprint: arXiv:1703.10354
9. Kaasbøll J, Berge O, Borge RE, Fjuk A, Holmboe C, Samuelsen T (2004) Learning object-oriented programming. In *Proceedings of the 16th Workshop of the Psychology of Programming Interest Group (PPIG 2004)*. Carlow, Ireland
10. Kitchenham B, Charters S (2007) *Guidelines for performing systematic literature reviews in software engineering* (EBSE Technical Report EBSE-2007-01). Keele University and Durham University
11. Liberman N, Beeri C, Kolikant B-D, Y (2011) Difficulties in learning inheritance and polymorphism. *ACM Trans Comput Educ* 11(1) Article 4. <https://doi.org/10.1145/1921607.1921611>
12. Lister R, Berglund A, Clear T, Bergin J, Garvin-Doxas K, Hanks B, Hitchner L, Luxton-Reilly A, Sanders K, Schulte C, Whalley JL (2006) Research perspectives on the objects-early debate. In *Working Group Reports on ITiCSE on Innovation and Technology in Computer Science Education* ACM. <https://doi.org/10.1145/1189136.1189183>
13. Luxton-Reilly A, Simon, Albluwi I, Becker BA, Giannakos M, Kumar AN, Ott L, Paterson J, Scott MJ, Sheard, J., Szabo C (2018) Introductory programming: A systematic literature review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education* (pp. 55–106). ACM. <https://doi.org/10.1145/3293881.3295779>
14. Moskal B, Lurie D, Cooper S (2004) Evaluating the effectiveness of a new instructional approach. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education* (pp. 75–79). ACM
15. Nelson HJ, Irwin G, Monarchi DE (1997) Journeys up the mountain: Different paths to learning object-oriented programming. *Acc Manage Inform Technol* 7(2):53–85. [https://doi.org/10.1016/S0959-8022\(96\)00024-0](https://doi.org/10.1016/S0959-8022(96)00024-0)
16. Page MJ, McKenzie JE, Bossuyt PM, Boutron I, Hoffmann TC, Mulrow CD, Shamseer L, Tetzlaff JM, Akl EA, Brennan SE, Chou R, Glanville J, Grimshaw JM, Hróbjartsson A, Lalu MM, Li T, Loder EW, Mayo-Wilson E,

- McDonald S, Moher D (2021) The PRISMA 2020 statement: An updated guideline for reporting systematic reviews. *BMJ* 372:n71. <https://doi.org/10.1136/bmj.n71>
17. Ragonis N, Ben-Ari M (2005) A long-term investigation of the comprehension of OOP concepts by novices. *Comput Sci Educ* 15(3):203–221. <https://doi.org/10.1080/08993400500224310>
 18. Ramalingam V, Wiedenbeck S (1997) An empirical study of novice program comprehension in the imperative and object-oriented styles. *Empirical Studies of Programmers: Seventh Workshop*. ACM, pp 124–137
 19. Reges S (2006) Back to basics in CS1 and CS2. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education* ACM. <https://doi.org/10.1145/1121341.1121432>
 20. Robins A, Rountree J, Rountree N (2003) Learning and teaching programming: A review and discussion. *Comput Sci Educ* 13(2):137–172. <https://doi.org/10.1076/csed.13.2.137.14200>
 21. Sanders K, Thomas L (2007) Checklists for grading object-oriented CS1 programs: Concepts and misconceptions. In *Proceedings of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '07)*. ACM
 22. Shmallo R, Ragonis N (2021) Understanding the this reference in object oriented programming: Misconceptions, conceptions, and teaching recommendations. *Educ Inform Technol* 26(1):733–762. <https://doi.org/10.1007/s10639-020-10265-6>
 23. Sorva J (2013) Notional machines and introductory programming education. *ACM Trans Comput Educ* 13(2). Article 8. <https://doi.org/10.1145/2483710.2483713>
 24. Sweller J (1988) Cognitive load during problem solving: Effects on learning. *Cogn Sci* 12(2):257–285. https://doi.org/10.1207/s15516709cog1202_4
 25. Sykes ER (2007) Determining the effectiveness of the 3D Alice programming environment at the computer science I level. *J Educational Comput Res* 36(2):223–244. <https://doi.org/10.2190/J175-Q735-1345-270M>
 26. Thomasson B, Ratcliffe M, Thomas L (2006) Identifying novice difficulties in object oriented design. In *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '06)*. ACM. <https://doi.org/10.1145/1140123.1140135>
 27. Underwood BJ (1957) Interference and forgetting. *Psychol Rev* 64(1):49–60. <https://doi.org/10.1037/h0044616>
 28. Uysal MP (2012) The effects of objects-first and objects-late methods on achievements of OOP learners. *J Softw Eng Appl* 5(10). <https://doi.org/10.4236/jsea.2012.510094>
 29. Vilner T, Zur E, Gal-Ezer J (2007) Fundamental concepts of CS1: Procedural vs. object oriented paradigm – a case study. *ACM SIGCSE Bull* 39(3). <https://doi.org/10.1145/1269900.1268835>
 30. White G, Sivitanides M (2005) Cognitive differences between procedural programming and object oriented programming. *Inf Technol Manage* 6:333–350. <https://doi.org/10.1007/s10799-005-3899-2>
 31. Wiedenbeck S, Ramalingam V, Sarasamma S, Corritore CL (1999) A comparison of the comprehension of object-oriented and procedural programs by novice programmers. *Interact Comput* 11(3):255–282. [https://doi.org/10.1016/S0953-5438\(98\)00029-0](https://doi.org/10.1016/S0953-5438(98)00029-0)
 32. Xinogalos S (2015) Object-oriented design and programming: An investigation of novices' conceptions on objects and classes. *ACM Trans Comput Educ* 15(3). Article 13. <https://doi.org/10.1145/2700519>
 33. Xinogalos S, Satratzemi M, Dagdilelis V (2006) An introduction to object-oriented programming with a didactic microworld: objectKarel. *Comput Educ* 47(2):148–171. <https://doi.org/10.1016/j.compedu.2004.09.005>

Figures

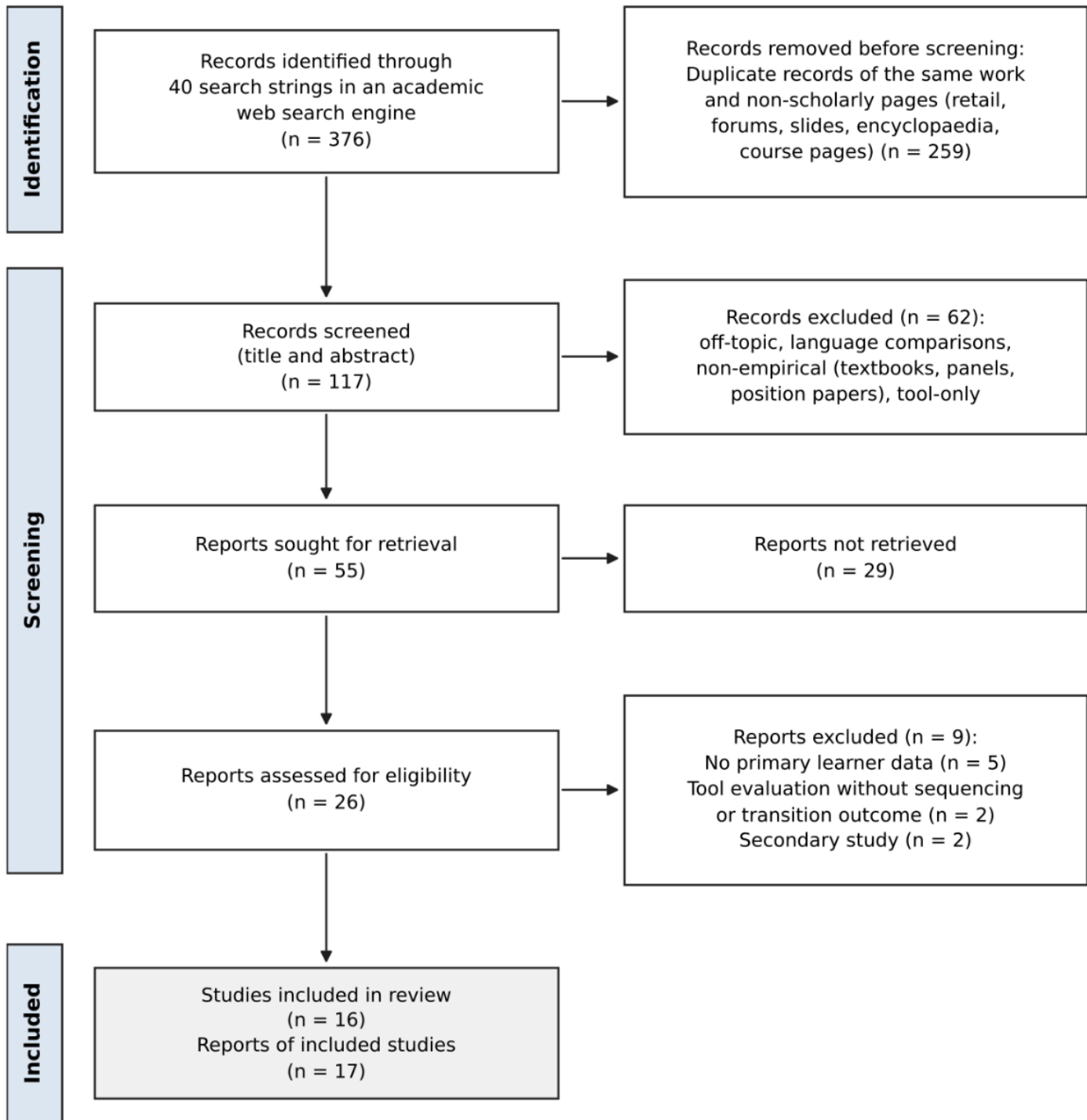


Figure 1

PRISMA 2020 flow diagram of the study selection process.