Using Design Alternatives to Learn About Data Organizations

Xingjian Gu Max A. Heller Stella Li Yanyan Ren Kathi Fisler Shriram Krishnamurthi
Contact:sk@cs.brown.edu
Computer Science Department
Brown University
Providence, RI, USA

ABSTRACT

Data that correspond to real-world scenarios can often be organized in several different ways in a database or program. Appreciating the differences between them and choosing an organization that addresses a system's needs are valuable and necessary computing skills. Unfortunately, little of the computing-education literature seems to deal with this topic.

In this paper we consider a technique for getting students to engage with this issue, grounded in theories of examples and differences. Instead of presenting a single organization, we present a pair of organizations and ask students to contrast them. Students then interact directly with the two organizations in a reflection step, which is followed by a further round of contrasting.

Our data show that even novice college students can handle this task fairly well. They are able to find many crucial differences (especially in terms of access and update operations), but also miss some (especially performance and privacy). These data suggest that this is a useful technique to pursue further, and also point to areas where students may need more instructional support.

CCS CONCEPTS

• **Information systems → Data structures; Database design and models**; • **Applied computing → Education**.

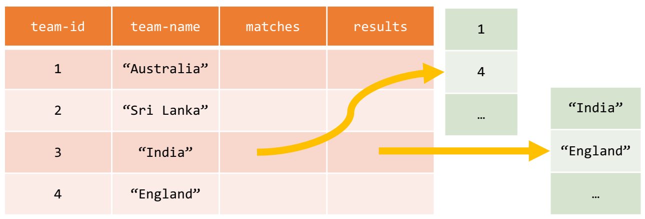
KEYWORDS

data structures; data organization design; introductory computing

ACM Reference Format:
Xingjian Gu Max A. Heller Stella Li Yanyan Ren Kathi Fisler Shriram Krishnamurthi. 2020. Using Design Alternatives to Learn About Data Organizations. In *Proceedings of the 2020 International Computing Education Research Conference (ICER '20), August 10–12, 2020, Virtual Event, New Zealand*. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3372782.3406267

1 INTRODUCTION

Imagine you are asked to design a system to capture information about a sports (cricket!) competition. On a little reflection, you realize there are multiple different ways to organize this information, which may be manifest as programming language data structures, database schemas, etc. You could choose an organization that prioritizes teams, such as this:

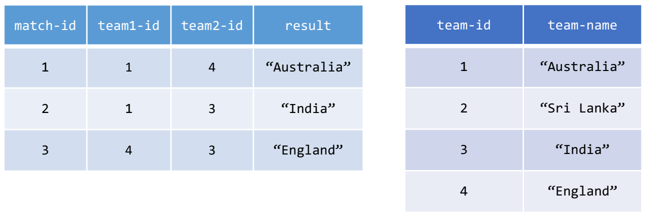


team-id	team-name	matches	results
1	"Australia"		
2	"Sri Lanka"		
3	"India"		
4	"England"		

1
4
…

"India"
"England"
…

Alternatively, you could focus on matches:



match-id	team1-id	team2-id	result
1	1	4	"Australia"
2	1	3	"India"
3	4	3	"England"

team-id	team-name
1	"Australia"
2	"Sri Lanka"
3	"India"
4	"England"

and so on. (We'll return to this pair in section 3.)

Making these kinds of trade-offs is a central concern in computer science and software engineering. It has both theoretical and practical consequences, including:

- the ability to express certain situations and hence to properly capture the domain being modeled;
- the efficiency of computations over the data (e.g., hierarchical representations can lead to better computational complexity over linear ones);
- the maintainability of the data (e.g., duplicating data can lead to updates creating inconsistencies);
- the complexity of the code that processes the data;
- the privacy of components of the data;

and more. Many of these issues are greatly accentuated when dealing with large, real-world data sets, as happens in data science.

In short, learning to organize data—in particular, to think through available design choices and understand their consequences—is a critical computing skill that students must develop. However, we find little attention paid to this problem in the computing-education research literature (section 2). It is even unclear what stage is developmentally appropriate for students to start exploring data organization design questions.

Process. In this paper, we ask whether novice college students can meaningfully start to tackle these questions. One natural approach would be top-down instruction. However, we wished to establish a baseline of what students could do with no instruction; we are also inspired by the theoretical basis on contrasting cases (section 2).

Our overall study structure is as follows:

(1) We give students a pair of data organizations for the same real-world setting, such as the earlier pair. We ask them for what trade-offs they see. This gives us insight into what set of concerns they initially think about.
(2) We then ask them to perform a reflective activity.
(3) We then ask them to repeat the process of describing trade-offs and providing justification.

The reader may well imagine several variants of this study; we discuss some of them in section 12.

Our research questions focus on whether students are able to make meaningful judgments at all, how the reflective step affects this, and what impact different styles of reflection have. The concrete research questions are described in section 4.3, after we have explained the process in more detail in section 4.2.

Contributions. This paper makes the following contributions:

- We introduce the problem of forming judgments about data organizations, which we believe has not gotten enough attention in the community (especially for novices).
- We present scenarios (section 3) that are accessible to novice collegiate programmers and yet contain enough complexity to lead to non-trivial reflections.
- We offer multiple rubrics (sections 5 and 8) for evaluating student responses that are applied across scenarios.
- We show that a (multi-step) process of reflection (section 4) is effective at getting students to meaningfully engage with the different organizations (sections 6, 7, 9 and 10), and learn about what students do and do not observe. The process (which we actively use in our classes) is not only a useful experimental construct but also valuable pedagogically.

Our findings are naturally contextual and conditional, and we discuss various threats in section 11.

Terminology: Data Structures versus Data Organization. We focus on data structures that represent real-world phenomena, not on classical "data structures" (such as queues, linked lists, or binary trees) that may be entirely artificial.

Consider two very different kinds of trees. A binary search tree is an abstract object, designed to improve performance. It is not designed to mirror a real-world counterpart. Thus, on the one hand, knowledge of the outside world does not dictate its design; on the other, it can support operations like balancing. In contrast, a family tree corresponds to a physical reality that constrains its shape; it cannot be "balanced". For lack of other terminology, we call these *computational* and *representational* data structures, respectively.

The focus of this paper is on representational data structures: ones that draw on a real-world context that even students without much computing knowledge can meaningfully reflect on. However, because the term "data structure" seems to be used extensively in the education literature to refer to the former kind (section 2), in this paper we use the phrase *data organization* consistently to mean representational data structures. There are many different ways in which these data can be organized (as we have already seen in section 1; likewise, a family tree could be ascending, descending, or not even formally a tree [11]). Such design alternatives are the focus of this paper.

2 THEORY AND RELATED WORK

Contrasting Cases and Variation. Our work is related to, and inspired by, a theory from perception. A classic paper [9] demonstrated that learning happens when noticing structural differences across examples that vary in a systematic way. Several later projects have built upon these ideas in a variety of fields (e.g., [3, 8]).

Our work is inspired even more closely by a series of papers, culminating in Schwartz, et al. [18], on *contrasting cases*. This work contrasts two instructional modes for physics: tell-and-practice, where students are given instruction in concepts and formulas first and then asked to practice it on a set of cases, and invent-with-contrasting-cases (ICC), where they are asked to derive the concepts and formulas from the cases. They find that both groups showed the same performance on word problems, but the ICC group had a better understanding of some physical phenomena, and did much better at transfer to a semantically-unrelated topic.

There are some differences between those works and ours. First, we consider only a pair of scenarios, not many in progression. Second, they often focus on the perception of low-level differences, whereas we are interested in the high-level *consequences* of the differences. Third, those papers focus on concrete entities (scribbles on a paper, chicks, etc.), whereas we give subjects data definitions, which are abstract. We are not aware of work in that field that addresses these differences. Though this theoretical basis is therefore not an exact fit for our work, we still employ lessons from it in the design of our research. We also speculate (section 12) on ways in which we might more tightly align our effort with that line of work.

Our work also relates to *variation theory* [15], which posits that learning occurs when someone perceives differences within a space, then constructs a generalization that captures its sameness. Our goal is to have students identify data-design trade-offs, not to discover a particular generalization about data organization.

Data Organization and Data Structures. The organizational questions we focus on sometimes manifest as programming language data structure specifications, and sometimes as schemata in external databases. These data organizations are also an important part of several program design methodologies that suggest program structures that follow data structures [5, 12].

However, this topic gets little coverage. The term "data structures" in the 2013 Curriculum Guidelines [13] appears to focus on computational data structures. Data organization is relegated to the "Information Management" elective under "Computational Science". Work on "basic" data-structure learning goals in CS2 [16], approaches to teaching data structures (e.g., [14, 19]), and difficulties students face (as captured in a concept inventory) [17], all focus on computational data structures. None of this is directly relevant to our work. Despite this marginalization, we believe that the rise of data science—and the focus on data even by those who will not major in computing—makes this topic deserving of our attention.

We have examined research on teaching software modeling (e.g., model-driven software engineering) and databases. Both demand attention to data organization (e.g., schemas). The papers we have found are either for upper-level students or discuss the topic generally without our specific focus. One notable exception [10] presents a classification of examples for teaching modeling.

A

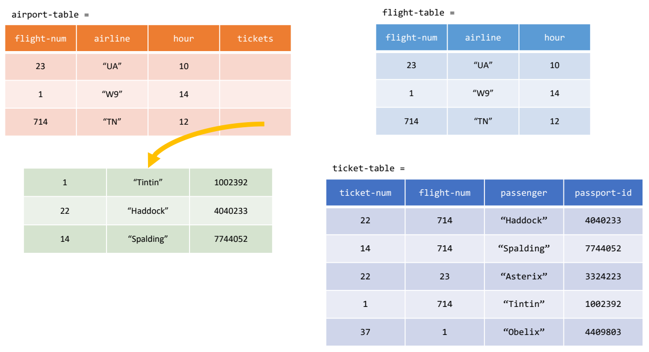
```
airport-table =
  table:
    flight-num :: Number,
    airline :: String,
    hour :: Number,
    tickets :: List<Ticket>
  end

data Ticket:
  | ticket(
      ticket-num :: Number,
      passenger :: String,
      passport-id :: Number)
end
```

B

```
flight-table =
  table:
    flight-num :: Number,
    airline :: String,
    hour :: Number
  end

ticket-table =
  table:
    ticket-num :: Number,
    flight-num :: Number,
    passenger :: String,
    passport-id :: Number
end
```



airport-table =

flight-num	airline	hour	tickets
23	"UA"	10	
1	"W9"	14	
714	"TN"	12	

1	"Tintin"	1002392
22	"Haddock"	4040233
14	"Spalding"	7744052

flight-table =

flight-num	airline	hour
23	"UA"	10
1	"W9"	14
714	"TN"	12

ticket-table =

ticket-num	flight-num	passenger	passport-id
22	714	"Haddock"	4040233
14	714	"Spalding"	7744052
22	23	"Asterix"	3324223
1	714	"Tintin"	1002392
37	1	"Obelix"	4409803

Figure 1: TRAVEL Data Organizations

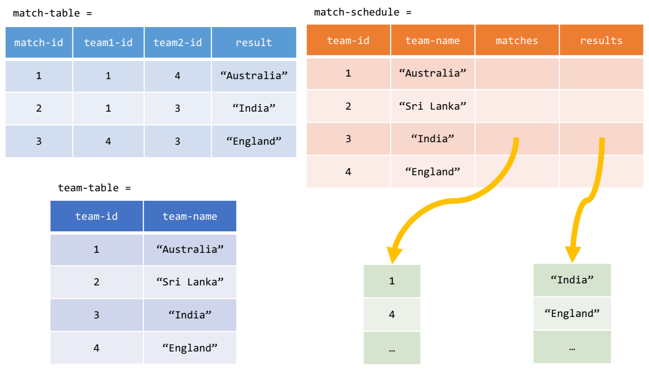
A

```
match-table =
  table:
    match-id :: Number,
    team1-id :: Number,
    team2-id :: Number,
    result :: String
  end

team-table =
  table:
    team-id :: Number,
    team-name :: String
  end
```

B

```
match-schedule =
  table:
    team-id :: Number,
    team-name :: String,
    matches :: List<Number>,
    results :: List<String>
  end
```



match-table =

match-id	team1-id	team2-id	result
1	1	4	"Australia"
2	1	3	"India"
3	4	3	"England"

team-table =

team-id	team-name
1	"Australia"
2	"Sri Lanka"
3	"India"
4	"England"

match-schedule =

team-id	team-name	matches	results
1	"Australia"		
2	"Sri Lanka"		
3	"India"		
4	"England"		

1
4
…

"India"
"England"
…

Figure 2: GAMES Data Organizations

3 STUDY SCENARIOS

We start by describing the data organizations that we used in the studies. There are two scenarios, which we call TRAVEL and GAMES. In each scenario, we consider two ways of organizing the data; that on the left we call "A" and the one on the right "B".

The data organizations are shown in fig. 1 and fig. 2. Students were shown *only* the textual definitions; the images are provided only to help the reader and were *not* part of the study. We expected students could produce concrete examples on their own, as both courses taught and graded students on this task from the outset.

The figures use the grammar of the Pyret language [1], but are hopefully understandable broadly. `table` creates a table, where each row has the comma-separated schema shown. Thus, for instance, `airport-table` is a table with four columns, the first of which is of numeric type representing the flight number, and so forth. `data` defines a new datatype; in this case, `Ticket` is a structure (akin to a struct in C or dataclass or dictionary in Python) with three fields (whose first field, for instance, is of numeric type representing the ticket number). In languages that don't support them natively, tables can simply be thought of as lists or arrays of structures. Therefore, these representations are straightforward to translate into languages with structures and sequences, from Python to Java to Haskell to OCaml and more (with types left out, as needed).

The TRAVEL *Organizations.* Both organizations can represent effectively the same information, i.e., a skilled programmer can transform an instance of one representation into an instance of the other without any loss. However, the two organizations provide two very different views of travel.

We urge the reader to pause here and formulate their thoughts on the two organizations before reading further.

A is oriented around airports: the primary structure is a table of flights. Each flight has a list of tickets, which contain details about individual travelers, but getting from an individual traveler's information (say their passport number) to the flights they are on requires some programming effort (that can be quite challenging for a novice, since it involves composing and nesting queries).

In contrast, B separates flights from tickets. Looking up a passenger does not require nesting because `ticket-table` is a top-level entity. However, this can impose a performance cost in multiple settings:

- Finding all the passengers on a particular flight is easy in A: search `airport-table` for the flight, and return the content of the corresponding `tickets` field. This is bounded by the number of flights. In contrast, in B this requires searching all of `ticket-table`, which is bounded by the total number of tickets.

- Finding information on a particular passenger on a specific flight is similarly impacted in B. In A, this requires searching only through the corresponding `tickets` field's list, which is bounded by the size of that plane. In B, this also requires searching all tickets.

At a high level, then, A often has better performance in return for programming effort overhead; B is vice versa.

The Games *Organizations.* The Games organizations, which we briefly saw in section 1, are also (almost: see section 12) equi-expressible, and have a somewhat parallel organization: one organization (A) has two tables like Travel:B, while the other (B) has one table like Travel:A. However, some of this similarly it superficial, because these have different characteristics.

We urge the reader to pause here and formulate their thoughts on the two organizations before reading further.

In Games:A, matches and teams are kept separate. The primary focus is on the games, which are in `match-table`. Looking up the names of the teams in a particular game requires queries into `team-table` (i.e., effectively a join).

In contrast, Games:B focuses on teams. For each team we have all its information, including a list of all its matches and results. This reduces the need for certain queries. However, it introduces a significant danger: data inconsistency. In A, information about a match is recorded only once. In contrast, in B, the information is stored once per team that competed in it. Thus a game can accidentally be recorded for one team but not the other, or be recorded for both but with different results. This also requires the two lists to be of the same length, with no mechanism for ensuring it. In short, there are many more subtle, unwritten, and (in mainstream languages) difficult-to-express invariants that govern the correct working of B.

4 STUDY METHODOLOGY

This paper describes two studies, each conducted in a different university course. The two had somewhat different protocols.

4.1 Study Contexts

Both classes were taught at a highly selective private university (Brown) in the USA. The classes had different professors (who are both co-authors of this paper). Both classes were taught primarily in a functional style. Both used Pyret.

CS-SumNon ("summer, non-major", CSCI 0050) is a course intended for non-majors and does not count towards the computer science major. About half the population of 21 students was actually in high school and had not yet been admitted to (any) university; the remainder were undergraduates at the university pursuing other majors. About 50% were female and about 10% minority.

The course was offered in the summer on a compressed schedule, so times below are mapped to the corresponding point in a 13-week US semester. The study was done as a homework assignment in the 9th-10th week of the semester. By then students had used tables for nearly eight weeks, and lists for two weeks.

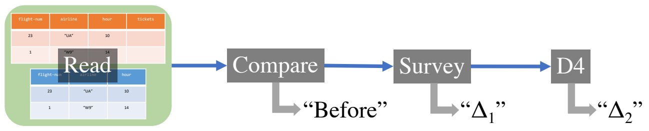
CS-AccInt ("accelerated introduction", CSCI 0190) is an accelerated introduction to computer science, offered over 13 weeks in the Fall semester. Students place in during the preceding summer by showing competency on lists and higher-order functions. (The placement process had used Racket [6].) Though many students have substantial high school computer science (and the course is designed primarily for them), about 10% of students have no real prior computer science; the summer readings are self-contained and assume no prior knowledge. The class had 59 students of whom (reflecting the diversity of high school computing courses) about 7% were female and about 7% were minority.

The study was conducted in a "lab": a two-hour session where about 20 students do assigned work. Students were graded on attendance, not work quality. The lab took place a month into the semester. By then students had extensive experience with lists. They were introduced to tables in the lab, where they first did some learning and practice tasks, but all students were familiar with lists-of-structures (which are roughly equivalent (section 3)), and several students had prior experience with tables directly. Different study conditions were assigned to different labs at random (each lab had just one condition, so all students could be given one URL to work from). Students picked labs based on their schedule.

Tools. The rest of this paper refers to two tools. D4 is a new, experimental tool that our group is developing for teaching data organization. The studies in this paper use D4's *directed* mode, in which students answer prompts by writing executable code (e.g., *write an example of data with feature Y*, *write an expression that computes X*). D4 checks answers for correctness against an instructor-provided predicate (that is specific to each question). The other tool, CPO (short for `code.pyret.org`), is the Web-based IDE for the Pyret language. It provides an editor and interactions window (REPL). Students use CPO in an *undirected* fashion: they generate ideas for executable code to try on their own, without prompts or feedback from checkers.

4.2 Study Protocol

CS-SumNon. Students worked only with the Travel scenario. Each student went through the following steps:



(1) They were given the two organizations and asked (free-response), *Compare and contrast the benefits of each organization. Are there scenarios or tasks for which one of the organizations seems better suited?* In the data analyses below, findings from this step are labeled "Before".

(2) They were then given the following specific scenarios (*directed*), in the form of a (static) survey, to spur thinking:
   (a) *A flight is considered "under-booked" if it has fewer than 50 tickets. Find all flight numbers for flights that are under-booked.*
   (b) *Given a ticket number, find the airline associated with its flight.*
   (c) *Determine if there are two tickets under the same name with different passport IDs.*
   (d) *Add a new field to every ticket that assigns a seat number to the passenger.*

Note that they were not asked to write any code; they were simply asked to consider the scenario, record whether they had a preference between the organizations just for that scenario (radiobox), and explain their thinking about the two organizations in light of it (free-response).

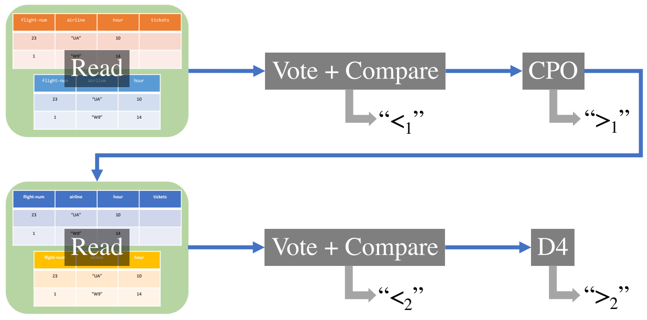
To keep the data manageable, we present an analysis that considers all four questions together, rather than individually. The data will show the *difference* to Before, labeled "Δ_1".

(3) They were then asked to work through more questions using D4 (also *directed*). They were shown concrete examples and asked questions on the same issues as above, but using these examples: e.g., *How many tickets are associated with the passport id 0195283?* and *Produce an updated* `ticket-table` *that reflects passenger "John" with passport number "9374544" purchasing ticket "8" for flight "8254".* The full instruments can be found at `https://cs.brown.edu/research/plt/dl/icer2020/`

(4) They were then asked the questions from step (2) again, followed by the question from step (1). The *additional* difference (to Δ_1) is labeled "Δ_2".

CS-AccInt. Students worked through both Travel and Games scenarios. Students in one group saw Travel first and Games second, while the other group saw Games first and Travel second. In addition, in each case, roughly half the students were shown the A and B organizations as listed above, while the other half were shown them in swapped order (B to the left, A to the right). These were normalized to the order shown for the purposes of analysis (including reinterpreting student answers on this basis). Because the total number of students is small, we do not check whether the spatial order of presentation had an impact.

The workflow is summarized in this graphic:



Concretely, each student went through these steps:

(1) They were shown the two organizations of one scenario and asked to: (1) Vote on which organization they preferred: A, B, Either, or Neither. (2) Compare and contrast the two. Their response is labeled "<".

(2) For the reflection step, instead of concrete questions, they were then put in CPO with the two definitions loaded; they were encouraged to experiment with them (*undirected*).

(3) They were asked to once again vote and to respond to the first question. The is labeled ">" and will show the *difference* to the immediately earlier < answer.

(4) They then repeated the above three steps for the other scenario, using D4 (*directed*) rather than CPO.

4.3 Research Questions

There are three main questions we set out to answer:

RQ 1 When asked to evaluate data organizations, what perspectives do students tend to take? What do they most focus on and what do they overlook?

RQ 2 How does the reflection step influence students' perceptions?

RQ 3 How does undirected reflection on organizations compare with directed reflection in terms of leading students to think about the design of data organizations?

To advertise where we are going: we find a surprising depth of student thought up front; we find the reflection results in noticeable improvements; we find some blind spots; and we also notice that some forms of directed reflection have little impact over undirected. Section 10 summarizes findings on these questions.

5 A RUBRIC FOR RESPONSES

For students' open-ended text, we used open-coding to develop a rubric. This is shown in fig. 3. The rubric is divided into high-level categories (like "Access"), each of which has one or more low-level tags (like "AEX"). It was developed by two of the authors. After six major iterations, the team obtained an inter-coder reliability of 0.926 using Cohen's κ. All subsequent data analysis was then conducted by one of the authors.

As in any such coding activity, student comments are going to directly or indirectly reflect details of their course. Different courses would possibly generate different responses to abstract over. Therefore, we do not claim these codes are universal. However: (a) they are useful for our analysis; (b) we believe they reflect fairly universal (though not all) concerns when discussing data organizations; and (c) they are a useful starting point for other researchers. We discuss this issue more in section 11 and section 12.

For readability, we have provided only abbreviated versions of the rubric. The exact rubric has much more fine-grained text (as needed to achieve a reasonable inter-coder reliability score). We give the full text at `https://cs.brown.edu/research/plt/dl/icer2020/`. (The same is true of the rubric in section 8.)

6 FINDINGS FROM CS-SUMNON

We begin by describing our findings for CS-SumNon. Recall (section 4.1) that these are non-major students and often still in high school. The table in fig. 4 summarizes our coding of their responses, with the columns as described in section 4.2.

We note that, without *any* prompting, a few categories immediately occur to students: most of them (naturally) think about how the organization will affect accessing data, but many also think about code structure for working with the data (such as nesting of queries).[1] Perhaps more surprisingly, they also think about the programmer experience. Updating, however, does not seem to occur to most of them (but they had not seen much of it in class, either).

The first reflection activity changes this significantly. Because the questions explicitly ask them to consider operations like extending the data, they naturally discuss updating. The operational nature of the reflection seems to also lead them to naturally describe

[1] Note that just because students wrote *about* some characteristic, it doesn't mean they did so *correctly*. We kept track of mistaken assertions in student responses. However, there were sufficiently few of these that we do not break them out for separate analysis.

Access Whether students consider how an organization impacts how data can be *accessed.*
- **AEX** Explains how a certain organization might make it easier or more difficult to perform tasks that involve accessing data from that organization.

Update How the data organization interacts with the difficulty of *making changes*. Answers that involve discussions about adding or removing data, or creating and removing the entire data organization, belong to this category.
- **UEX** Explains how a certain organization might make it easier or more difficult to perform tasks that involve updating data from that organization.

Algorithm Efficiency Whether students discuss storage space or runtime *efficiency* issues. Students were given the tag even if they did not use notations like big-O in their description.
- **SC** Shows awareness of the space it takes to store data for either organization.
- **TM** Mentions the running time of algorithms.

Example Whether students can conceptualize how the different data organizations could be used, and see if they can think of *examples*, such as scenarios, cases, or programming tasks, in which either organization might perform better than the other.
- **NRL** Has *no* mentions of real life applications, but instead solely discuss the organization abstractly. The percentage of instances of this in our dataset rounds to 0.
- **CDE** Considers coding tasks, implementation of certain features, or the functions needed to achieve a certain goal.

Miscellaneous *Higher-level discussions* about the data organizations, such as their general design, whether the organization is intuitive for either end-users or developers, and how easy they are to learn and use.
- **SP** Security and privacy issues.
- **PEX** Programmer experience: focuses on how the design of the organization influences a programmer's experience when working with it.
- **DTC** Data-type choices: what data types were chosen to represent data in the table (e.g., numbers and strings) and how that matters.

External Issues Things that are not directly related to students' answers to the assignment itself, but rather some external factors that could have influenced students' performance. We ignore these in this paper.

Figure 3: Response Rubric

programming consequences. However, some categories (like efficiency and privacy) neither occur to them originally nor as a result of answering these questions. This is perhaps unsurprising for an introductory non-majors course, but these data also suggest areas that demand more explicit instruction. (While some may consider privacy issues outside the scope of an introductory course, others might argue that, for non-majors who will use that course to do basic data science in other disciplines, it is imperative to expose them to this concern.)

The low occurrence of the time and space efficiency tags is not surprising. Being a course for non-majors, the course does not cover big-O and had not talked about performance at the point of the study. Past research [7] shows, however, that even when a course does not mention it at all, students pick up these issues (perhaps from teaching assistants, fellow students, parents, etc.), which might explain the non-zero occurrences.

The second reflection activity uses D4. Here, we see very little impact. In most cases this is unsurprising: either the category was already discussed by everyone, or it was essentially outside the scope of D4's prompts (such as privacy). We do see some additional impact on thinking about the programmer experience.

Category	Tag	Before	Δ_1	Δ_2
Access	AEX	74%	26%	0%
Update	UEX	26%	74%	0%
Alg. Eff.	TM	11%	0%	0%
	SC	5%	5%	0%
Example	CDE	53%	47%	0%
	SP	0%	0%	0%
Misc.	PEX	53%	11%	16%
	DTC	21%	0%	5%

Figure 4: Categories and Tags for CS-SumNon **(N=21)**

	Travel	Games
Changed Preference	60%	52%
No Change	40%	48%
Initially liked:		
A	33%	41%
B	31%	28%
Either	29%	22%
Neither	7%	9%
Eventually liked:		
A	21%	53%
B	60%	17%
Either	19%	22%
Neither	0%	7%

Figure 5: Preference Votes for CS-AccInt **(N=59)**

7 FINDINGS FROM CS-ACCINT: PART 1

As we can see from section 6, prompting students with very specific scenarios right away results in them quickly considering several standard issues. However, this is both unrealistic in general (when they are designing a new data organization, they need to be able to generate prompts for themselves) and does not give us a useful baseline for how they would do in an unstructured setting. This is why the protocol for CS-AccInt changes (section 4.2) to the much more realistic setting of simply giving them an IDE in which to interact with the definitions and arrive at their own conclusions. For the second scenario, we gave them D4 instead of CPO. This was both

Category	Tag	Travel <	Travel >	Games <	Games >
Access	AEX	90%	9%	86%	10%
Update	UEX	21%	19%	24%	31%
Alg. Eff.	TM	14%	7%	7%	3%
	SC	7%	3%	19%	5%
Example	CDE	50%	22%	48%	19%
Misc.	SP	5%	0%	0%	2%
	PEX	38%	24%	48%	16%
	DTC	2%	0%	7%	5%

Figure 6: Categories and Tags for CS-AccInt (N=59)

to compare the effects, and to limit time: we imagined undirected reflection could take much longer than answering focused questions with sample data.

First, we look at how students voted their preference and how their votes changed. Recall (section 3) our own (relatively expert) view: We found true trade-offs between Travel:A and Travel:B, noting that Travel:A provided better performance in some common cases, but this came at the cost of greater code complexity. In contrast, we claimed that Games:A was significantly better than Games:B, due to the latter's much greater likelihood of error.

Figure 5 summarizes student opinion before and after reflection. In Travel, students initially come down quite evenly between the two.[2] However, after reflection, they seem to significantly prefer B. In Games, they begin with a slight preference for A, and end with a much stronger preference for it.

Irrespective of their alignment with our view, these data suggest that students were actively choosing (rather than taking the easy way out with "Either"), and were also seriously re-assessing their views in light of the reflection activity. Furthermore, from the written remarks, we see ample evidence of students engaging deeply with the activity.

Going into this study, we expected that this population would do significantly better than that of CS-SumNon. These students typically had much more computer science experience, including (in many cases) multiple years of high school computer science; the content of CS-AccInt was also much more sophisticated. By the time of this lab, they had already spent two weeks working with big-O, including having to submit a graded big-O analysis (and many had prior high school experience with it).

Figure 6 shows our coding of the student responses to the open-ended questions.[3] For simplicity, here, we ignore whether they saw that scenario first or second, grouping both positions together. We dive into this distinction later, in section 10 (where we will not see big differences).

Despite the differences in student population, we were surprised by how similar the "Before" column of fig. 4 is to the "Travel <" and "Games <" columns of fig. 6. If anything, the CS-SumNon students seem to be a little more analytic (lacking evidence, we choose not to speculate on reasons why). Because the sample sizes are relatively small we do not believe there will be much use in running formal statistical analyses of differences; indeed, we are not claiming (based on our reading of student comments) that there *are* significant differences between the two populations for a test to tease apart. We also noticed that the CS-AccInt audience did not use big-O very much. This was somewhat surprising: not only were they learning it and starting to use it in assignments, some had already seen it earlier in high school, suggesting that lack of familiarity was not the issue: perhaps it did not occur to them instinctively.

Same Preference Only applied when students did not change their preference between pre-exercise and post-exercise answers.
- **REIT** Reiteration: reflection did *not* stimulate them to think about something they had not considered before.
- **PROT** Praise the other: after reflection, they saw advantages to the organization they did not prefer.
- **STR** Strengthen: affirms their pre-reflection answer with new, concrete ideas, or provided justification or reasoning to support their decision to not change their preference.

Changed Preference Only applied when students changed their preference after reflection.
- **LRN** Learned more: learned or discovered aspects of the data organization that they had not considered before the exercise.
- **CRCT** Correction: explicitly mentions that what they thought before doing the exercise turned out to be false, and they corrected their belief after doing the exercise. This tag is narrower than LRN, in the sense that change in preference is not only caused by new realizations, but that the student found something that contradicts with their prior beliefs about the data organizations when doing the exercise.

Changed Reasoning Applied if the student approached the evaluation from the same perspective (either concrete coding aspects or abstract high-level aspects) but had new understanding or new ideas about them. Applies whether or not students changed preference.
- **EXCD** Expanded coding ideas: student saw coding with the data organization in a new light, and changed their opinion regarding how they would code with that organization.
- **EXNC** Expanded non-coding ideas: student saw the high-level or abstract aspects of the data organization in a new light, and changed their opinion about them.

Proposed Improvements Students' answers proposed improvements.
- **ORG** Proposed changes to the data organization.
- **FEAT** Proposed to add new features.

Figure 7: Difference Rubric

[2]Our analysis of written comments indicates that students generally did not make a clear distinction between "Either" and "Neither", and most of the latter were really closer to the former. Therefore, these two entries can essentially be summed and treated as "Either".

[3]Rubric tag ratios for CS-AccInt are relative to the number of students in each condition, as opposed to the total number of students in the class.

		Travel	Games
Same Pref.	REIT	14%	12%
	PROT	24%	21%
	STR	47%	43%
Ch. Pref.	LRN	29%	41%
	CRCT	12%	5%
Ch. Reas.	EXCD	43%	38%
	EXNC	24%	21%

Figure 8: Difference Categories and Tags for CS-AccInt

8 A RUBRIC FOR DIFFERENCES

An easy way to spot differences in student thinking after reflection is to just look at the set of tags. If a new tag appears after reflection, this suggests some impact on their thinking. (The absence of a tag is far less meaningful: students may simply not feel like repeating themselves. Therefore, we did not consider "dropped" tags.)

However, this is only a coarse-grained reflection of their change in opinion. Furthermore, even in cases where the tags remain the same, in CS-AccInt we were able to identify qualitative differences in their written answers. We therefore created a separate rubric to assess *differences* (shown in abbreviated form in fig. 7). This rubric, too, has categories with corresponding tags. It was created by the same two coders and deployed by the same one, and after three major revisions arrived at a Cohen's κ of 0.716.

9 FINDINGS FROM CS-ACCINT: PART 2

The second part of our analysis for CS-AccInt therefore applies the difference rubric (section 8). The results are shown in fig. 8.

We find, in particular, that students exhibit all the behaviors we would hope to see: through reflection, they (a) learned to better appreciate the organizations after their initial vote (PROT, LRN), (b) learned about aspects of their preferred organization that they had not considered earlier (STR), (c) corrected some of their views (CRCT), and (d) changed their reasoning (EXCD, EXNC).

Naturally, these tags are well-represented because the rubric was created as a consequence of seeing such comments with some frequency. However, these also seem to represent a good set of considerations we would hope to see student engage with in reflecting over different data organizations.

10 RESEARCH QUESTION RESPONSES

Based on all these data, we now return to the original research questions that framed our study and try to address them. Note that the answers to **RQ 1** and **RQ 2** only repeat and summarize information from earlier in the paper; **RQ 3** does that but also presents new data.

RQ 1*. When asked to evaluate data organizations, what perspectives do students tend to take? What do they most focus on and what do they overlook?*

In general, we find that students focus on low-level but important operations such as accessing parts of the structure, and less so on updating it (unless prompted). Program efficiency, whether formal (in terms of big-O) or informal, is not a major consideration when contrasting organizations, but the programmer experience (which is a "human efficiency" consideration) is. Code examples are routinely used to express their thoughts. Perhaps unsurprisingly, privacy issues are rarely mentioned. (They would not distinguish between these organizations anyway.) We find it interesting that these issues are raised in similar proportions by accelerated collegiate students and a population of non-majors and pre-college (under 18) students ("Before" in fig. 4 versus "Travel <" and "Games <" columns of fig. 6).

		Travel	Games
AEX	cpo	8%	16%
	d4	9%	4%
UEX	cpo	23%	41%
	d4	16%	19%
TM	cpo	12%	3%
	d4	3%	4%
SC	cpo	4%	9%
	d4	3%	0%
CDE	cpo	12%	22%
	d4	31%	15%
SP	cpo	0%	3%
	d4	0%	0%
PEX	cpo	15%	16%
	d4	31%	15%
DTC	cpo	0%	9%
	d4	0%	0%

Figure 9: Rubric Tag Gains for CS-AccInt

RQ 2*. How does the reflection step influence students' perceptions?*

We see this step having a large impact in both study settings. We notice it in the set of issues that students raise, and even more starkly in their change in preferences. Indeed, we found it necessary to create a rubric of differences to capture the broad set of issues that arose when comparing their answers before and after reflection.

RQ 3*. How does undirected reflection on organizations compare with directed reflection in terms of leading students to think about the design of data organizations?*

We expected directed reflection to have much stronger outcomes than undirected. Our data suggest much more ambiguity.

Students saw two forms of directed reflection. In CS-SumNon, we see a huge impact from the initial survey, as shown in the Δ_1 column of fig. 4. They also use d4 after that, which has a much smaller impact, but this is unsurprising because the d4 prompts are very similar to those of the survey.

In CS-AccInt, the first reflection step was undirected: they were just given cpo to play with. For their second scenario, they used d4 for directed reflection. Recall that students who did Travel first (with cpo) did Games second (with d4) and vice versa. The data in fig. 6 does not distinguish between the two. Here, we delve into it.

Figure 9 shows the *gain* in tags after the reflection step. For each tag, there is an entry for each of cpo (first reflection) and d4 (second). The numbers shown are the ratio of students who were in that category who added that tag after reflection. As each of these reflections was on a different scenario, this serves as a

		Travel	Games
REIT	CPO	15%	0%
	D4	13%	27%
PROT	CPO	27%	19%
	D4	22%	23%
STR	CPO	46%	44%
	D4	47%	42%
LRN	CPO	35%	50%
	D4	25%	31%
CRCT	CPO	12%	3%
	D4	13%	8%
EXCD	CPO	50%	34%
	D4	38%	42%
EXNC	CPO	31%	22%
	D4	19%	19%

Figure 10: Difference Rubric Tag Gains for CS-AccInt

formative comparison between letting students freely explore the data organizations (CPO, undirected) and guiding their explorations with explicit prompts (D4, directed).

The data show that there is not much increase in value to using D4 with CS-AccInt (and indeed, the raw change ratios are sometimes lower). We suspect a few possible factors:

- It may be that the prompting that D4 offers is either uninteresting or simply redundant for the CS-AccInt audience, as opposed to the value directed reflection has for CS-SumNon. It would be interesting to contrast using D4 with just a simple, static, non-interactive form (as given to CS-SumNon).
- From reading responses, we detected a certain degree of fatigue in the second part. Whether D4 was the cause of the fatigue or the victim of it cannot be established with the information we have, but it would indicate why there was less depth of change.
- The CS-AccInt audience is sophisticated enough that they may have seen some similarities between the two activities, and hence were less engaged in the second scenario (i.e., instead of fatigue, the cause may be ennui).

Recall that we introduced a rubric of differences to give us more fine-grained information than the initial rubric's tags can on their own. In fig. 8, we saw the rate of use of the tags in the difference rubric. Again, however, we did not distinguish between what kind of reflection step had occurred that led to that difference. In fig. 10 we separate these by CPO versus D4. Again, for the most part, these do not reveal any major advantage to using D4, and suggest D4 underperformed compared to just CPO. (The one place where there is a stark difference that appears to be in D4's favor—27% to 0%—is REIT, which is merely reiteration without enhancement (fig. 7).)

11 THREATS TO VALIDITY

11.1 Internal Validity

Because our students are novices, we provided sample organizations to consider instead of asking them to generate these. However, it is possible that they might have generated different organizations than ours, which they would have found much more "natural" than ours. Providing ours instead may have inhibited the quality of their work on the organizations, rendering their responses poisoned by the choices we made.

Coding (as in classification, not programming) inherently contains biases of omission, commission, and blind spots. Though we have used inter-coder reliability, this only tells us that the coders have similar biases, not that all biases have been eliminated. Nevertheless, we believe the codes we have arrived at represent a reasonable set of concerns about data organization, and thus that our rubrics still have some value despite containing latent biases.

Student-preference changes after directed reflection—the two steps in CS-SumNon and D4 in CS-AccInt—are very likely to be highly influenced by the prompts they saw. These in turn largely reflect the biases of the designers of those prompts: namely us. Therefore, student responses (and hence the rubrics) may well be biased by the kinds of issues we ourselves tend to most consider as developers and educators.

11.2 External and Ecological Validity

Our rubrics not only still contain potential biases, as noted in section 5, but are likely to also be shaped significantly by course content. We have argued that the rubrics still appear to describe fairly universal concepts (if not all of them) in discussing data structures. If this is incorrect (e.g., if we suffer from confirmation bias), then the rubrics are not of much value in other settings, nor can an analysis based on them usefully be compared against those by others. Only through more research (by others) in this area can we determine whether and to what extent this is a problem.

A significant assumption behind our work is that students can reason about abstract data definitions—most probably by constructing concrete example data for themselves. This is a skill that takes some practice. As noted in section 3, our study classes heavily emphasize writing tests for programs, and (following HtDP [5]) even recommend writing examples before starting to work on a solution. Therefore, from early on, these students learn to construct data examples. Students without that facility—for instance, those who are not required to do high-quality testing—may struggle to envision concrete data examples and may thus have difficulty reasoning through the effect of various operations.

Programming language style could also be a significant factor. Both CS-SumNon and CS-AccInt primarily use functional programming, in which updates to data are much less common than in imperative programming. As a result, our participants may have thought about access much more than update.

11.3 Construct Validity

By giving students a pair of organizations, we give them a perhaps valuable tool for reasoning about data. However, we have not provided evidence that, left to themselves, they are capable of coming up with such pairs (or more). We suspect they can (section 12), but if they in fact cannot, then this work is of limited value for when these students are faced with later programming tasks.

When prompted, our students did engage in meaningful analysis. However, this does not imply that they would necessarily pause to prompt themselves. A more open-ended study would leave them unprompted and see whether they even consider organizational alternatives, which ones, and what decisions they make. It is also unclear what differences we would see between novice and experienced programmers in this regard; we suspect some of the literature on schema retrieval [2] would almost certainly apply here too.

A broader concern is the causal relationship between this exercise and building a system. Ultimately, one of our main goals is to make students into better programmers (in later courses, as data scientists, as professional developers, etc.). If, however, their analyses of the data organizations do not lead to actionable change, then arguably the exercise has not had much value. Evaluating this requires seeing the effect of their analysis on their overall software-development process.

12 DISCUSSION

Alternative and Follow-Up Studies. The particular protocol used in this study (section 4.2) naturally generates ideas for several variations:

- Given a prose description of a real-world setting, can novices generate even one data organization?
- Given a prose description of a real-world setting, can novices generate more than one data organization? There is some evidence that with a little instruction, they can produce different multiple different program plans [4]. Does this carry over to data?
- What is the role of pictures as opposed to textual representations of concrete data?
- What would the quality of responses be if we showed students only one data organization instead of two? That is, how essential is the *contrast* to idea generation?
- If we give students two organizations that are not equi-expressive, do they detect it? Can they write concrete examples of situations that one organization can represent but the other cannot? (Note that, depending on interpretation, the Games organizations may not be exactly equi-expressible. We haven't indicated what `match-id` means. Does it give an ordering of the matches? If so, Games:A has information that Games:B does not. Either way, Games:B does not have a `match-id`. None of our students mentioned this.)
- As time passes, applications add support for new operations and sunset some older ones. Because a data organization is often chosen based on expectations of operations and their frequency, changes to them require revising the organization. How well can students perform these changes?

The Role of d4. Even though d4 was not the focus of this paper, we were somewhat surprised that it did not do better than undirected reflection in CS-AccInt. As noted in section 10, student responses suggested a certain degree of fatigue or ennui when answering questions about the second scenario, where d4 was deployed.

A natural follow-up is to alternate reflection activities so that some students see d4 for the first scenario. Care must be taken with experiment design for two reasons. First, students would already have extensive experience with their ide (like cpo), unlike d4. Second, d4 prompts in the first scenario may implicitly carry over to the second one (e.g., getting them to think about updates). Both issues reduce the purity of the comparison.

Presenting Data Organizations. We have presented the data organizations in section 3 using *algebraic datatypes*, which are commonly found in functional languages and increasingly in hybrid languages that borrow from functional programming, such as Scala and TypeScript. This notation arguably lays out the data organization very crisply to the initiated. In contrast, the mixing of data and code in object-orientation may make the underlying data organization much harder to see in code; it may benefit from, or even require, a presentation like a uml diagram for the reader to clearly see the structure. Adapting this work to other linguistic settings therefore clearly needs some thoughtful effort.

Transfer. The literature on contrasting cases suggests that it is very useful for transfer [20]. Transfer was not a focus of our study, but naturally, the purpose of doing exercises such as ours is to help students develop transferable skills in analyzing data organizations. Even before addressing far-transfer, it would be valuable to use scenarios that are superficially different (e.g., different domains) but similar in the abstract, to judge whether students are able to near-transfer their judgment between the scenarios.

Belief Revision. From a pedagogic perspective, the data in this paper show that students engage deeply with design alternatives. This suggests that they are well prepared for subsequent classroom instruction on the topic. However, there is a small danger that they may have gotten overly attached to their views and may be unreceptive to expert analyses of the same structures. Exactly how much bottom-up experimentation is healthy so that students form initial impressions and are ready to engage in top-down instruction without becoming entrenched remains an open question for this kind of work.

Computational Thinking. What is the role of data reasoning in computational thinking? Some formulations of the term emphasize algorithms and procedural thinking, but in the process overlook data organization. However, the structure of data has major consequences for system behavior—and, as we show with preliminary evidence, is something even novices can reason about. Notably (as their written answers demonstrate), they are able to pass judgment on the data using examples of real-world situations and uses that they likely could not themselves program.

We therefore believe activities like the one described in this paper can lead to a richer, fuller formulation of computational thinking, and can generate new and engaging activities that many students can participate in. Indeed, computing background does not seem to matter much in our data, and a more diverse group of students is likely to make a much richer set of judgments—thereby enabling students in traditionally underrepresented groups to contribute based on their unique experiences, and also feel more valued.

ACKNOWLEDGMENTS

K. Ann Renninger introduced us to contrasting cases. Mark Guzdial and Jan Vahrenhold engaged in useful conversations on terminology. We thank the reviewers for their comments and related work references. This work was partially supported by US NSF grants.

REFERENCES

[1] [n.d.]. Pyret Programming Language. `https://www.pyret.org/`. Accessed: 2020-04-10.

[2] Beth Adelson. 1981. Problem solving and the development of abstract categories in programming languages. *Memory & Cognition* 9, 4 (1981), 422–433.

[3] Irving Biederman and Margaret M. Shiffrar. 1987. Sexing Day-Old Chicks: A Case Study and Expert Systems Analysis of a Difficult Perceptual-Learning Task. *Journal of Experimental Psychology: Learning, Memory, and Cognition* 13, 4 (1987), 640–645.

[4] Francisco Enrique Vicente Castro, Shriram Krishnamurthi, and Kathi Fisler. 2017. The Impact of a Single Lecture on Program Plans in First-Year CS. In *Koli Calling International Conference on Computing Education Research*.

[5] Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, and Shriram Krishnamurthi. 2001. *How to Design Programs*. MIT Press. `http://www.htdp.org/`

[6] Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, Shriram Krishnamurthi, Eli Barzilay, Jay McCarthy, and Sam Tobin-Hochstadt. 2018. A Programmable Programming Language. In *Communications of the ACM*.

[7] Kathi Fisler, Shriram Krishnamurthi, and Janet Siegmund. 2016. Modernizing Plan-Composition Studies. In *ACM Technical Symposium on Computer Science Education*.

[8] Eleanor J. Gibson. 1969. *Principles of Perceptual Learning and Development*. Appleton-Century-Crofts.

[9] James J. Gibson and Eleanor J. Gibson. 1955. Perceptual Learning: Differentiation or Enrichment? *Psychological Review* 62, 1 (1955), 32–41.

[10] Martin Gogolla and Antonio Vallecillo. 2012. On Explaining Modeling Principles with Modeling Examples: A Classification Catalog. In *Educators' Symposium of the MODELS Conference*.

[11] Partick Höse. 2011. Cycles in family tree software. `https://stackoverflow.com/questions/6163683/cycles-in-family-tree-software`. Accessed: 2020-04-10.

[12] Michael A. Jackson. 1975. *Principles of Program Design*. Academic Press.

[13] Joint Task Force on Computing Curricula. 2013. Computer Science Curricula 2013. `https://www.acm.org/binaries/content/assets/education/cs2013_web_final.pdf` Accessed: 2020-04-10.

[14] Raymond Lister, Ilona Box, Briana Morrison, Josh Tenenberg, and D. Suzanne Westbrook. 2004. The dimensions of variation in the teaching of data structures. *SIGCSE Bulletin* 36, 3 (June 2004).

[15] Ference Marton. 2015. *Necessary conditions of learning*. Routledge, New York.

[16] Leo Porter, Daniel Zingaro, Cynthia Lee, Cynthia Taylor, Kevin C. Webb, and Michael Clancy. 2018. Developing Course-Level Learning Goals for Basic Data Structures in CS2. In *ACM Technical Symposium on Computer Science Education*.

[17] Leo Porter, Daniel Zingaro, Soohyun Nam Liao, Cynthia Taylor, Kevin C. Webb, Cynthia Lee, and Michael Clancy. 2019. BDSI: A Validated Concept Inventory for Basic Data Structures. In *SIGCSE International Computing Education Research Conference*.

[18] Daniel L. Schwartz, Catherine C. Chase, Marily A. Oppezzo, and Doris B. Chin. 2011. Practicing versus inventing with contrasting cases: The effects of telling first on learning and transfer. *Journal of Educational Psychology* 103, 4 (2011), 759–775.

[19] Josh Tenenberg. 2003. A Framework Approach to Teaching Data Structures. In *ACM Technical Symposium on Computer Science Education*.

[20] Edward L. Thorndike and Robert S. Woolworth. 1901. The influence of improvement in one mental function upon the efficiency of other functions. *Psychological Review* 8 (1901).