Play Your Cards Right: Using Quantitative Card-Sort Data to Examine Students' Pattern-Like Concepts

James Finnie-Ansley
The University of Auckland
Auckland, New Zealand
james.finnie-ansley@auckland.ac.nz

Paul Denny
The University of Auckland
Auckland, New Zealand
paul@cs.auckland.ac.nz

Andrew Luxton-Reilly
The University of Auckland
Auckland, New Zealand
andrew@cs.auckland.ac.nz

ABSTRACT

In order to transfer problem solving knowledge effectively across tasks, students must be able to identify when two problems can be solved in similar ways. As instructors, it is helpful to know how well students can identify such similarities. Unfortunately, current efforts to understand how students categorise similar problems are laborious and time-consuming, involving intensive data collection and analysis. Our own prior work on analysing student categorisations of simple algorithmic patterns involved 35 hour-long interviews and time-consuming open coding of the resulting transcripts. It is clear that traditional approaches to elicit such knowledge are not scalable or reproducible.

We propose a method of automatically analysing student classification of simple problems which examines digitally collected quantitative data from online card sorting exercises. We compare the results of this automated analysis with our own prior work which examined qualitative data and find that both analyses reveal similar insights. Research in this area is new; however, quantitative analysis of card sorting data provides an effective and scalable method of quickly identifying how students view problem similarities. The method proposed in this paper seems promising for a wide range of topics and exercises including formative and summative assessment.

CCS CONCEPTS

• **Social and professional topics** → **Computing education**.

KEYWORDS

card sorting, problem similarity, patterns, problem statements, hierarchical clustering

ACM Reference Format:

James Finnie-Ansley, Paul Denny, and Andrew Luxton-Reilly. 2022. Play Your Cards Right: Using Quantitative Card-Sort Data to Examine Students' Pattern-Like Concepts. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V: 1 (SIGCSE 2022)*, March 3–5, 2022, Providence, RI, USA. ACM, New York, NY, USA, 7 pages. <https://doi.org/10.1145/3478431.3499343>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SIGCSE 2022, March 3–5, 2022, Providence, RI, USA

© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-9070-5/22/03...\$15.00

<https://doi.org/10.1145/3478431.3499343>

1 INTRODUCTION

Constructivism is a theory of learning that states learning involves building on the motivations and existing knowledge of learners, and that meaning is constructed relative to learners' existing knowledge through experience [6, 20]. One implication of constructivism for computing education research (CER) is that we must first know what students already understand before we can build upon that understanding [7, p. 19-20]. We have long since known education research, in general, needs to move beyond the mere identification and replacement of misconceptions and should focus on *conceptions* [39]; however, much of CER still focuses on misconceptions [11, 13]. Such a focus has been argued to influence discourses on knowledge and disregard the productive knowledge of learners, even in works that claim to be focusing on conceptions [13]. There has been work in CER which explores different conceptions of novice and naive learners such as the 'commonsense computing' series [9, 22, 23, 29, 37, 38, 45] which explored naive students' understanding of different computing concepts pre-instruction; however, it is argued establishing new discourses on productive knowledge in CER is an uphill and ongoing battle [13].

One barrier to understanding the productive knowledge of students is the often intensive data collection and analysis required. For example, all the 'commonsense computing' works required reading through and coding student answers or interview transcripts. Our own prior work which examined students' concepts of simple algorithmic patterns involved 35 hour-long interviews and coding of the resulting transcripts [17]. Such intensive data collection and analysis is not scalable or readily reproducible.

In this work, we build upon a prior study and propose a scalable and automatic method of evaluating students' concept structure using quantitative analysis. We compare findings derived from this new method with that of our prior work which analysed only qualitative data from interviews. The quantitative approach is fully automatable and scalable, so can be applied in large classes, but also permits the rapid production of individual feedback and is thus suitable for formative and summative assessment activities.

To explore the applicability of the proposed method we answer the following research question:

RQ1. To what extent can quantitative analysis of card sort data provide insight into student conceptions when compared with traditional methods of qualitative analysis?

To answer this research question, we evaluate card sorting data with item analysis (quantitative analysis) and compare this with the results of prior work which evaluated sorts with category-based analysis (qualitative analysis). The card sorting exercises were performed with 35 participants that span the breadth of four years of tertiary-level study. Participants were asked to group cards

which contained simple algorithmic problem statements (similar to those described in [30]) in two rounds. In the first round, each participant was asked to group cards based on what *intuitively* felt meaningful to them as a programmer. In the second round, they were asked to group cards based on how they would *solve* each problem. In this paper, we present the item analysis of this data and evaluate the viability of this analysis as an alternative to traditional qualitative approaches such as category analysis.

2 RELATED WORK

2.1 Patterns

Patterns were introduced by Alexander in their book “A Pattern Language” [1] which focused on common problems and their solutions in architecture, urban design, and community livability. The book defined a new language, a *pattern language*, outlining several patterns which describe a problem and a solution to that problem often with references to other patterns forming a network. Alexander’s work on patterns was then adapted and explored in computer science [5, 18] and became popular after the publication of “Design Patterns” by Gamma et al. with the development of *software design patterns* [12]. Patterns have since been extended to cover problems across multiple levels of abstraction from architecture patterns [19] to elementary design patterns [46]. It is argued patterns should be integrated across instruction with *pattern-oriented instruction*, establishing *pattern thinking* as a core element of problem solving [42]. It is also suggested patterns should be assimilated in stages and tailored to students’ current ability [25, 30, 32].

While patterns have gained attention in CER, by themselves they are not sufficient for instruction on programming or program design. Patterns are still promising for instruction as they are a tangible schema-like representation of programming problem concepts, abstracting the messy and interconnected domain of programming.

2.2 Card Sorting

Card sorting is a knowledge elicitation technique to reveal the structure of knowledge and the categorisation a participant applies to a topic [15, 34]. Card sorting has been explored in computing domains to elicit programmers’ knowledge structures on a range of computing concepts. For example, two sorting studies by Lewandowski et al. and by Murphy et al. examined novice and senior students’ abilities to group cards with programming concepts on them (such as “function” or “scope”) [24, 31]. These studies found that novice and less advanced students struggled to meaningfully group abstract concepts but were more confident with concrete concepts which are colloquially discussed during instruction; whereas more advanced senior students were able to sort cards based on wider abstract criteria. Card sorting has also been used to examine expert perceptions of the relevance of different patterns and the order patterns should be taught across tertiary level instruction [25]. Card sorting is also commonly used in requirements engineering [27], human-computer interaction [44], and internet architecture [41].

Several card sorting studies in computing use card sorts where each card contains a single word representing a computer science concept (e.g. [24, 31, 35]) or code fragments (e.g. [14]). Our study differs in that we use cards which only include question statements as the prompts. A card sorting exercise similar to the one presented

in this paper examined novice and expert knowledge representations in physics [10] finding novice physics students focused on surface features of problems whereas experts focused on underlying problem features. This work was later extended to examine interventions in physics education which have shown short-term improvements in physics students’ categorisation criteria [16].

3 METHOD

3.1 Participants

Study participants were recruited from programming-focused computer science and software engineering courses across the first four years of their respective academic programmes at the University of Auckland (UoA). This is a large, urban, research-focused University in New Zealand. Students at UoA typically take a CS1 course which teaches objects late and focuses on procedural programming with many small automatically assessed programming exercises (similar in style to the many small programs approach described by Allen et al. [2]) using an open source automated assessment tool¹ using the Python programming language. Students in CS1 courses at UoA are not given explicit instruction on plans, simple patterns (such as elementary [46], loop [3], or algorithmic [30] patterns), or similar constructs; but, solve problems which would be within the scope of problems covered by such patterns.

The study was advertised to students through course announcements, at each year level, four weeks before the end of the academic year in 2020. Participation in the study was voluntary and students were compensated for their time, which was estimated to be around one hour, with a \$50 voucher. A total of 35 students responded to our invitation to take part in the study. Of these, we have data for 32 participants (9 women and 23 men) who completed both the card sort and follow-up interviews. Participants reported on their current grade point average and the programming language(s) they were most familiar with. Table 1 summarises the cohort level, median self-reported grade point average, and self-reported primary programming language for these participants.

Cohort	N	Median GPA	Primary Language	N
Beginner	16	A	C#	1
			Python	15
Intermediate	9	A-	Java	2
			Python	6
			Python & JavaScript	1
Advanced	7	A-	Java	1
			JavaScript	1
			JavaScript & Java	3
			Python	2

Table 1: Participant Distribution

¹Coderunner [26]: <https://coderunner.org.nz/>

3.2 Materials

3.2.1 Tools. Based on feedback obtained during a short pilot study, Trello² was selected for conducting the card sorts. Other tools tested in the pilot were Optimal Sort³, and a traditional physical card sort.

3.2.2 Questions. The pilot study also confirmed 30 questions to be an appropriate number of questions for the card sorting exercises – a quantity which is recommended as a minimum number of cards to include in a card sort [41]. A set of 30 questions was generated loosely oriented around variations of the rainfall problem. The rainfall problem was introduced by Soloway as a variation of an earlier ‘averaging problem’ [40]. The rainfall problem has become somewhat of an informal benchmark for beginner programmers across the CER community and has been used in several studies over several decades [33, 36].

The problems involved in this study were of similar difficulty to the rainfall problem; however, care was taken to control the scope and variation of the problems generated. Students participating in this study will have solved problems of similar difficulty and scope to the problem set used in this study during their CS1 courses at UoA, but the problems used in this study are not identical to the problems given to students as exercises. The following examples are questions used verbatim in the card sorting exercises and will be referred to when describing the problem constraints which follow:

Example 3.1. Write a function which takes a list of integers as input and returns a new list containing all the integers from the input list which are greater than 5.

Example 3.2. A rainfall sensor records the daily rainfall in mm. A day with over 100mm of rainfall is considered as having “heavy rainfall”. Write a function which takes the daily rainfall data you have collected as a list of integers. The function should return a new list containing the rainfall data for the days with heavy rainfall.

Example 3.3. Write a function which takes a list of strings as input and returns the first string which begins with a lowercase letter. If no such string exists, return an empty string.

The constraints applied when generating problems are discussed briefly below.

Scope. All the problems involved in the card sorting exercises are well-structured (see [21]) and ask the solver to write a function which takes a list as input and returns some kind of result. The original list is never modified and some kind of implicit or explicit linear iteration is needed to solve each problem. Ensuring all problems are of this form limits potential superordinate contrast which is recommended for card sorting exercises [28]. For example if non-linear recursion questions were also included, the superordinate categories *iteration* and *recursion* would be contrasting. The constraint of one list parameter and no in-place modifications also limits contrast between questions.

Questions also only vary in three major ways: patterns, levels of contextualisation, and list element type. Such variation allows

for problems which have known similar and dissimilar surface features (contextualisation and element type) and underlying features (patterns).

Each problem would only require one pattern to solve. This was because including multiple patterns in a question would make the questions harder and their categorisation more ambiguous as participants would need to consider issues such as pattern merging (as described by Soloway [40]).

Patterns. Five separate problem types were chosen, each of which require the same pattern to solve regardless of variations in context and element type. The chosen problem types were: Filtering, Mapping, Averaging/Summing, Linear Searching, and Extreme Values. For example, Example 3.1 and Example 3.2 are both *filtering* problems and Example 3.3 is a linear search problem. While patterns are considered an underlying feature of problem statements, as was found in a study by Docktor et al. [16] involving physics problems, underlying features can sometimes reveal themselves through language. This was unavoidable in many instances, particularly for Averaging/Summing (using words such as ‘average’, ‘mean’, ‘sum’), Extreme Values (using words such as ‘maximum’), and Searching (using words such as ‘find’).

Contextualisation. Problems were constructed to have one of two types of context: *computing only* context, or *mixed* computing and real-world context. *Computing only* context refers to problems which only contained references to computing entities in their statements. *Mixed* context refers to problems which contained references to both ‘real world’ and computing entities in their statements. Example 3.1 and Example 3.3 are both computing only context questions, and Example 3.2 is mixed context.

Mixed context questions in this study either used the example of a rainfall sensor gathering daily rainfall data or paragraph analysis where the solver is described as wanting to analyse words in a paragraph. Functions in rainfall questions all accepted a list of integers as a parameter and paragraph analysis questions all accepted a list of strings as a parameter.

Types. List element types were either numbers (ints and floats) or Strings. These types were selected as they are built-in immutable types which are effectively primitive (taught as their own data-type typically before objects or other composite types) in Python and Java, which are the core languages used in the programming courses at UoA.

Sentence Structure. Problems followed similar sentence and problem structure. All computing context problems loosely followed the following structure:

Write a function which takes a list of {strings, integers, floats} as input and returns {a new list containing (property of input list), (property of input list)}.

Where ‘property of input list’ refers to something which can be computed from the input list. For example, all the elements greater than 5, or each element multiplied by 2, the average, etc.

Mixed context problems loosely followed the following structure:

{Context setup}. Write a function which takes {real world data} as a list of {strings, integers, floats}. The

²A web-based list-making application by Atlassian: <https://trello.com/>

³An online card sorting and analysis tool by Optimal Workshop: <https://www.optimalworkshop.com/optimalsort/>

function should return {a new list containing ⟨property of real world data⟩, ⟨property of real world data⟩}.

Where ‘context setup’ refers to a short question preamble explaining the problem context. ‘Property of real world data’ is similar to the ‘property of input list’ described before, but refers only to real world entities such as the days with heavy rainfall etc.

Problem goals for mixed context questions were also only explained in relation to real-world entities but a two directional mapping from real-world entities to function inputs was given. This ensures mixed context questions are well-structured and that the real-world component of problem statements is integral to comprehension of the question. For example, Example 3.2 describes the goal in terms of real world entities (heavy rainfall) but provides a mapping (*daily rainfall data* $\leftrightarrow$ *integer list*) which can then be used to map the real-world goal into a computing context.

3.3.3 Question Distribution. The set of 30 questions used in the card sorting exercises were approximately equally split between patterns, context, and input type. The Cartesian product of the patterns, level of contextualisation, and input type was taken and two questions were written from every resulting set. From this pool of 40 questions, 30 were selected for the card sort. Questions could be uniquely identified with a tag as an identifier to help with understanding groupings of questions when analysing results. Tags are three to four characters long. The first character refers to *pattern*: (F)iltering, (M)apping, (A)veraging/Summing, (S)earching, and (E)xtreme Values. The second character refers to *context*: (C)omputing or (M)ixed context. The third character refers to *input type*: (S)tring or (N)umber. An optional fourth character refers to a duplicate question type, as two questions of each combination of pattern, context, and input was written, this was merely so each question could be uniquely identified from its tag. This character was just a “1” at the end of the tag. For example, the tag “SCN” refers to a searching question, that is computing only context, and takes a list of numbers as input.

3.3 Procedure

The general organisation of the experiment with each participant is described in the following sections. After each of the card sorting exercises, participants answered questions about the overall rationale of their card sorts in a semi-structured interview. As this is not a focus of the data reported here details of this have not been included. For more information on the interviews, see [17].

3.3.1 Introduction to Trello and test sort. After participants gave consent and discussed their programming experience, they were introduced to Trello and its controls including how to create a new list, rename a list, move cards between lists, scroll horizontally, and remove empty lists. Participants were then shown a test board which included four preloaded addition and subtraction questions in a list called ‘Cards’ and were asked to sort the cards in whatever way they liked to get familiar with Trello’s controls.

3.3.2 Card Sort 1. Participants were then shown the instructions for the first sorting exercise. These were displayed on the monitor the participants would be using to do the sort and read aloud by the researcher. The instructions for the first sort asked participants to group cards in a way which was *meaningful to them* as

a programmer. This was elaborated on through an analogy to the differences in how a car mechanic and the researcher (someone who knows nothing about cars) might group cars in a car yard. These instructions were intended to capture how the participants might “intuitively” group the different problems.

If the participant had no questions about the instructions, the first card sort would begin. The participants were then shown a Trello board with 30 preloaded questions in a list called “Cards” similar to the test sort and were told to begin sorting when they were ready.

3.3.3 Card Sort 2 Instructions and Worked Example. Before the second sort began, participants were shown a new set of instructions and given a worked example demonstrating the thinking process participants should have for the second sort. As before these were displayed on the monitor the participants were using for the sorts and read aloud by the researcher. For the second sort students were asked to group cards based on how they would *solve the problems* described in the problem statements. Participants were shown a card sorting exercise with four cards on a demonstration Trello board and the researcher would ‘think aloud’ while sorting them into groups based on their solutions. The cards in the worked example were two counting questions and two averaging/summing questions.

If the participant had no questions about the worked example or the second card sort, the second card sort would begin. This card sort began the same as the first with a Trello board with a single ‘Cards’ list on the left of the screen with 30 programming problems on individual cards.

4 RESULTS

This study focuses on an item analysis technique which examines groupings of cards without examining names or descriptions (see [15]). The dissimilarity matrix was taken between cards for all participants for Sort 1 and Sort 2. The dissimilarity matrix measures how often each pair of cards are *not* put together⁴. For example, a distance of 0 between two cards means every participant put those two cards together in a group, and a distance of 30 means 30 of the participants did **not** put those cards together in a group (See [43]).

Hierarchical clustering can be used on the dissimilarity matrix to give an overall view of what cards are viewed as most similar by participants. For our analysis, we used complete linkage of the Euclidean distances to produce the hierarchical clustering. Figure 1 shows the dendrograms of the hierarchical clustering for Sorts 1 and 2 respectively.

The leaves of these dendrograms represent questions used in the card sorting exercises. Hierarchical clusters give an overall idea of the ‘average’ card sort performed by participants. Branches of the dendrogram join lower down if the groups of cards at their leaves are more similar.

The dendrogram for Sort 2 has much tighter groups corresponding with particular patterns (the first letter of each tag) with each of these groups joining much lower. This suggests for Sort 2, pattern-based groups are tightly bound and separate from other patterns.

⁴A similar method of measuring distance between cards is the *similarity* matrix which is the opposite of a *dissimilarity* matrix. For an example of a similarity matrix, see [4].

Figure 1: Hierarchical Cluster Dendrograms

However, the branches of Sort 1 show no obvious pattern, with several branches converging higher up the dendrogram. This suggests that there is no overall pattern for the first sort and cards are not overall seen as similar to participants as a group.

4.1 Edit Distance

Hierarchical clusters can be ‘split’ into k groups by cutting the dendrogram horizontally across k branches. This allows us to use metrics such as edit distance to evaluate these averaged sorts. We took the edit distance of each averaged sort to several *probe sorts* which are sorts constructed by researchers to examine a criterion of interest.

The edit distance between two sorts is the minimum number of card moves needed to transform one card sort into the other (see [15] for a more detailed description). It can be used as a metric of similarity between two card sorts; the lower the edit distance, the more similar the sorts. While we have used edit distance to evaluate the aggregate of participant card sorts, edit distance can also be used to evaluate individual participant’s sorts.

We constructed several probe sorts by sorting cards according to pattern, context, and input. Dendrograms were cut into the same number of groups as the probe sort they were being compared to. For example, the pattern-based probe sort has 5 groups, so each of the dendrograms were cut into five groups and compared with this probe. Table 2 shows the distance to various probe sorts.

An obvious pattern emerges from the edit distances for Sort 1 and Sort 2. For the first sort, where participants intuitively grouped

cards, the edit distance to the input-type probe is 0, meaning the dendrogram for Sort 1 split into two groups is equivalent to sorting cards based on input type. Indeed, this can be seen in the dendrogram for Sort 1; splitting the dendrogram in two (cutting across the top two branches) perfectly splits the questions that take lists of strings as input (tagged xxS) on the left and questions that take lists of numbers as input (tagged xxN) on the right. However, the height at which these branches converge suggests that this relation is weak. For the second sort, where participants were prompted to focus on solutions to problems, the lowest edit distance is to the pattern-based probe sort with a distance of 1. Again, this can be seen in the dendrogram; cutting the dendrogram for sort two into 5 branches divides questions by pattern (the first character of each tag) except for question MMN which has been grouped with filtering. In this case, the groups join much lower down on the dendrogram suggesting strong relationships between questions in each group. This indicates for Sort 1, participants tended to group cards according to input type even though this relation is weak and for Sort 2, participants grouped cards strongly according to pattern.

These results are in line with the results of our previous study, which only examined qualitative data gathered from group names and interviews after each card sort. During each card sorting exercise, participants were aware they needed to name each of the groups they created and after each card sorting exercise, participants described each of these groups. Our previous study only examined group names and descriptions which were treated as a single unit, and each unit was coded into one of 18 unique categories which emerged during an open coding process. Each of these 18 categories could be considered part of broader “surface”, “intermediary”, and “underlying” categories. With both input type and context being considered as surface level and patterns being considered as underlying. Our previous study found:

For all of the ‘underlying’ features, we observe a positive change between Sort 1 and Sort 2 – more students were able to identify underlying features in the second sort... Almost all of the surface features had a

Probe Type	Groups	Sort 1	Sort 2
Pattern	5	18	1
Context	3	9	17
Input	2	0	14

Table 2: Edit Distance to Probe Sorts

corresponding negative change indicating fewer participants sorted according to surface features. [17]

5 DISCUSSION

The analysis of card sorting data presented in this paper is preliminary — only looking at the edit distance between aggregated sorts to probe sorts. Other analyses such as multidimensional scaling [43] of both cards and participants or factor analysis [8] may be possible with larger sample sizes providing richer insight into student conceptions.

Despite being rather simple, the quantitative analysis provides similar insight into student conceptions as the qualitative analysis. Both were able to show a general shift away from surface features towards underlying features between Sorts 1 and 2. The hierarchical clustering also provides insight into how tight the categories participants create are, with the second dendrogram showing the groups merge low down indicating the cards within those groups are perceived as being close together.

While the quantitative analysis has shown to provide a good overall view of the card sorting data, it is limited in terms of its power to examine or explain student categorisation. Our previous study was interested in examining not only the categories students created but the *language* participants used to describe those categories. By analysing group names and descriptions we were able to examine the level of specificity students used when categorising problems, student ability to describe pattern-based concepts using established pattern language, and student justifications for certain modes of categorisation. This type of analysis is not possible with the item-based analysis described here. Despite these drawbacks, the automatically collected data and automatic analysis provides a scalable way of evaluating the categorisation of large cohorts both in aggregate and individually by student – something the qualitative analysis cannot do.

6 IMPLICATIONS FOR PRACTICE AND FUTURE WORK

We believe card sorting exercises and item-based analysis of card sorting exercises show promise for both research and practice. While card sorting has been used in computing education research to examine student understanding of computing terms, the results of this study and our prior analysis show that card sorting can also be used to examine student understanding of abstract problem concepts (i.e. patterns) and automatic item analysis can reveal trends in student understanding of these concepts and provide simple metrics for understanding them.

The card sorts in this study were open sorts, where participants were not given any predefined categories into which cards should be sorted. Open sorts are typically done for exploratory studies where the categorisation space is not known. There may be some difficulty in using open sorts for formative and summative assessment; in such cases, closed card sorts may be preferable and more work will be needed to assess their viability.

Potential future work is to address the implementation and setup of card sorting exercises and to evaluate their performance in authentic learning contexts. We have developed a set of resources to support card sorting exercises. This includes a tool that parses and

analyses results from card sorting data, an example set of questions, and a method for constructing questions. We have made these resources available to the community as a contribution of the current work⁵. We would be interested in exploring the applicability of card sorting exercises in practice and working with practitioners on evaluating card sorting exercises in authentic contexts.

7 LIMITATIONS

While automatic analysis of card sorting data has proven useful in eliciting concept structure from students, care needs to be taken when constructing card sorts. In the construction of this study, extreme care was taken to ensure questions used in the card sorts were unambiguous in their categorisation along a particular axis (e.g. pattern, context, and input). This restriction limited the scope of problems that could be included as it prevented complex questions that required the application of multiple patterns. For example, a question that asks participants to average the non-negative integers in a list would be inappropriate as it is unclear how the question would be categorised (is it a filtering question, an aggregating question, or does it belong in a separate category?). Because of this, it is unclear whether *pattern merging* [40] would be able to be unambiguously tested using card sorting methods. Some methods of card sorting, such as sorts allowing the duplication of cards or more advanced analyses such as factor analysis may be able to handle such complex questions; however, more work is needed to explore the applicability of card sorting for eliciting complex concept structure.

8 CONCLUSIONS

Automatic quantitative analysis of card sorting data has shown to reveal the same overall insights into students' understanding of simple algorithmic patterns as a time-consuming qualitative analysis. Some information cannot be readily ascertained from the item analysis described here, such as the language and justifications participants provide for their sorts. However, the ease of data collection and analysis, and scalability of this method of performing card sorts makes it a viable method of assessing large cohorts both in aggregate and individually.

We must first know what students already understand before we can build upon that understanding. The traditional focus within computing education research on the mere identification and replacement of misconceptions is not enough and new discourses on productive knowledge are needed. One difficulty with a focus on productive knowledge is the often intensive data collection and analysis required to understand it. This work proposes an automatic and scalable method of understanding students' positive conceptions of abstract problems. Using card sorting question types and quantitative analysis as a method of assessing student understanding of abstract problem concepts is a new and unexplored area. However, the promising results from a simple and preliminary quantitative analysis give us reason to believe this is a viable method of assessing and understanding student knowledge.

⁵These resources can be found at the following link: <https://github.com/James-Ansley/trellocardsortparser>

REFERENCES

- [1] Christopher Alexander. 1977. *A pattern language: towns, buildings, construction*. Oxford university press.
- [2] Joe Michael Allen, Frank Vahid, Alex Edgcomb, Kelly Downey, and Kris Miller. 2019. An Analysis of Using Many Small Programs in CS1. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) (SIGCSE '19). Association for Computing Machinery, New York, NY, USA, 585–591. <https://doi.org/10.1145/3287324.3287466>
- [3] Owen Astrachan and Eugene Wallingford. 1998. Loop patterns. In *Proceedings of the Fifth Pattern Languages of Programs Conference*.
- [4] Kathy Baxter, Catherine Courage, and Kelly Caine. 2015. Chapter 11 - Card Sorting. In *Understanding your Users (Second Edition)* (second edition ed.), Kathy Baxter, Catherine Courage, and Kelly Caine (Eds.). Morgan Kaufmann, Boston, 302–337. <https://doi.org/10.1016/B978-0-12-800232-2.00011-0>
- [5] Kent Beck and Ward Cunningham. 1987. Using pattern languages for object-oriented programs. <http://c2.com/doc/oopsla87.html> (1987).
- [6] Mordechai Ben-Ari. 2001. Constructivism in computer science education. *Journal of Computers in Mathematics and Science Teaching* 20, 1 (2001), 45–73.
- [7] John Bransford, John D Bransford, Ann L Brown, and Rodney R Cocking. 1999. *How people learn: Brain, mind, experience, and school*. National Academies Press.
- [8] Miranda G. Capra. 2005. Factor Analysis of Card Sort Data: An Alternative to Hierarchical Cluster Analysis. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 49, 5 (2005), 691–695. <https://doi.org/10.1177/154193120504900512> arXiv:<https://doi.org/10.1177/154193120504900512>
- [9] Tzu-Yi Chen, Gary Lewandowski, Robert McCartney, Kate Sanders, and Beth Simon. 2007. Commonsense Computing: Using Student Sorting Abilities to Improve Instruction. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education* (Covington, Kentucky, USA) (SIGCSE '07). Association for Computing Machinery, New York, NY, USA, 276–280. <https://doi.org/10.1145/1227310.1227408>
- [10] Michelene T H Chi, Paul J Feltoch, and Robert Glaser. 1981. Categorization and representation of physics problems by experts and novices. *Cognitive science* 5, 2 (1981), 121–152.
- [11] Michael Clancy. 2004. Misconceptions and attitudes that interfere with learning to program. *Computer science education research* (2004), 85–100.
- [12] Michael J Clancy and Marcia C Linn. 1999. Patterns and pedagogy. *ACM SIGCSE Bulletin* 31, 1 (1999), 37–42.
- [13] Brian A Danielak. 2019. Deprecating Misconceptions through Context-Dependent Accounts of Productive Knowledge. In *Proceedings of the 2019 ACM Conference on International Computing Education Research* (ICER '19). Association for Computing Machinery, New York, NY, USA, 91–100. <https://doi.org/10.1145/3291279.3339424>
- [14] Simon P. Davies, David J. Gilmore, and Thomas R.G. Green. 1995. Are Objects That Important? Effects of Expertise and Familiarity on Classification of Object-Oriented Code. *Human-Computer Interaction* 10, 2-3 (1995), 227–248. <https://doi.org/10.1080/07370024.1995.9667218>
- [15] Katherine Deibel, Richard Anderson, and Ruth Anderson. 2005. Using edit distance to analyze card sorts. *Expert Systems* 22 (2005), 129–138. <https://doi.org/10.1111/j.1468-0394.2005.00304.x>
- [16] Jennifer L Docktor, José P Mestre, and Brian H Ross. 2012. Impact of a short intervention on novices' categorization criteria. *Physical review special topics-physics education research* 8, 2 (2012), 20102.
- [17] James Finnie-Ansley, Paul Denny, and Andrew Luxton-Reilly. 2021. A Semblance of Similarity: Student Categorisation of Simple Algorithmic Problem Statements. In *Proceedings of the 17th ACM Conference on International Computing Education Research*. Association for Computing Machinery, New York, NY, USA, 198–212. <https://doi.org/10.1145/3446871.3469745>
- [18] E Gamma, R Helm, R Johnson, and J Vlissides. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education.
- [19] Ian Gorton. 2006. *Essential software architecture*. Springer Science & Business Media.
- [20] Kenneth Howe and Jason Berv. 2000. Constructing constructivism, epistemological and pedagogical. *Teachers College Record* 102, 7 (2000), 19–40.
- [21] David H Jonassen. 1997. Instructional design models for well-structured and ill-structured problem-solving learning outcomes. *Educational technology research and development* 45, 1 (1997), 65–94.
- [22] Gary Lewandowski, Dennis J Bouvier, Tzu-Yi Chen, Robert McCartney, Kate Sanders, Beth Simon, and Tammy VanDeGrift. 2010. Commonsense understanding of concurrency: computing students and concert tickets. *Commun. ACM* 53, 7 (2010), 60–70.
- [23] Gary Lewandowski, Dennis J. Bouvier, Robert McCartney, Kate Sanders, and Beth Simon. 2007. Commonsense Computing (Episode 3): Concurrency and Concert Tickets. In *Proceedings of the Third International Workshop on Computing Education Research* (Atlanta, Georgia, USA) (ICER '07). Association for Computing Machinery, New York, NY, USA, 133–144. <https://doi.org/10.1145/1288580.1288598>
- [24] Gary Lewandowski, Alicia Gutschow, Robert McCartney, Kate Sanders, and Dermot Shinnars-Kennedy. 2005. What Novice Programmers Don't Know. In *Proceedings of the First International Workshop on Computing Education Research* (Seattle, WA, USA) (ICER '05). Association for Computing Machinery, New York, NY, USA, 1–12. <https://doi.org/10.1145/1089786.1089787>
- [25] Tracy L Lewis, Mary Beth Rosson, and Manuel A Pérez-Quinones. 2004. What Do The Experts Say? teaching introductory design from an expert's perspective. *ACM SIGCSE Bulletin* 36, 1 (2004), 296–300.
- [26] Richard Lobb and Jenny Harlow. 2016. Coderunner: A tool for assessing computer programming skills. *ACM Inroads* 7, 1 (2016), 47–51.
- [27] Neil A M Maiden and M Hare. 1998. Problem domain categories in requirements engineering. *International Journal of Human-Computer Studies* 49, 3 (1998), 281–304.
- [28] Jean M Mandler and Patricia J Bauer. 1988. The cradle of categorization: Is the basic level basic? *Cognitive development* 3, 3 (1988), 247–264.
- [29] Robert McCartney, Dennis J. Bouvier, Tzu-Yi Chen, Gary Lewandowski, Kate Sanders, Beth Simon, and Tammy VanDeGrift. 2009. Commonsense Computing (Episode 5): Algorithm Efficiency and Balloon Testing. In *Proceedings of the Fifth International Workshop on Computing Education Research Workshop* (Berkeley, CA, USA) (ICER '09). Association for Computing Machinery, New York, NY, USA, 51–62. <https://doi.org/10.1145/1584322.1584330>
- [30] Orna Muller, David Ginat, and Bruria Haberman. 2007. Pattern-Oriented Instruction and Its Influence on Problem Decomposition and Solution Construction. *SIGCSE Bull.* 39, 3 (jun 2007), 151–155. <https://doi.org/10.1145/1269900.1268830>
- [31] Laurie Murphy, Renee McCauley, Suzanne Westbrook, Timothy Fossum, Susan Haller, Briana B Morrison, Brad Richards, Kate Sanders, Carol Zander, and Ruth E Anderson. 2005. A multi-institutional investigation of computer science seniors' knowledge of programming concepts. *ACM SIGCSE Bulletin* 37, 1 (2005), 510–514.
- [32] Nelishia Pillay. 2010. Teaching design patterns. In *Proceedings of the SACLAC conference, Pretoria, South Africa*.
- [33] Anthony V Robins. 2019. Novice programmers and introductory programming. *The Cambridge handbook of computing education research* 1 (2019), 327–376.
- [34] Gordon Rugg and Peter McGeorge. 2005. The sorting techniques: a tutorial paper on card sorts, picture sorts and item sorts. *Expert Systems* 22, 3 (2005), 94–107.
- [35] Kate Sanders, Sally Fincher, Dennis Bouvier, Gary Lewandowski, Briana Morrison, Laurie Murphy, Marian Petre, Brad Richards, Josh Tenenber, Lynda Thomas, Richard Anderson, Ruth Anderson, Sue Fitzgerald, Alicia Gutschow, Susan Haller, Raymond Lister, Renée McCauley, John McTaggart, Christine Prasad, Terry Scott, Dermot Shinnars-Kennedy, Suzanne Westbrook, and Carol Zander. 2005. A multi-institutional, multinational study of programming concepts using card sort data. *Expert Systems* 22, 3 (2005), 121–128. <https://doi.org/10.1111/j.1468-0394.2005.00303.x>
- [36] Otto Seppälä, Petri Ihtola, Essi Isohanni, Juha Sorva, and Arto Vihavainen. 2015. Do We Know How Difficult the Rainfall Problem Is?. In *Proceedings of the 15th Koli Calling Conference on Computing Education Research* (Koli, Finland) (Koli Calling '15). Association for Computing Machinery, New York, NY, USA, 87–96. <https://doi.org/10.1145/2828959.2828963>
- [37] Beth Simon, Dennis Bouvier, Tzu-Yi Chen, Gary Lewandowski, Robert McCartney, and Kate Sanders. 2008. Common sense computing (episode 4): Debugging. *Computer Science Education* 18, 2 (2008), 117–133.
- [38] Beth Simon, Tzu-Yi Chen, Gary Lewandowski, Robert McCartney, and Kate Sanders. 2006. Commonsense Computing: What Students Know before We Teach (Episode 1: Sorting). In *Proceedings of the Second International Workshop on Computing Education Research* (Canterbury, United Kingdom) (ICER '06). Association for Computing Machinery, New York, NY, USA, 29–40. <https://doi.org/10.1145/1151588.1151594>
- [39] John P Smith III, Andrea A DiSessa, and Jeremy Roschelle. 1994. Misconceptions reconceived: A constructivist analysis of knowledge in transition. *The journal of the learning sciences* 3, 2 (1994), 115–163.
- [40] Elliot Soloway. 1986. Learning to program= learning to construct mechanisms and explanations. *Commun. ACM* 29, 9 (1986), 850–858.
- [41] Donna Spencer. 2009. *Card sorting: Designing usable categories*. Rosenfeld Media.
- [42] Asher Sterkin. 2006. Teaching Design Patterns. In *FECS*. Citeseer, 167–176.
- [43] Tom Tullis and Bill Albert. 2013. Chapter 9 - Special Topics. In *Measuring the User Experience (Second Edition)* (second ed ed.), Tom Tullis and Bill Albert (Eds.). Morgan Kaufmann, Boston, 209–236. <https://doi.org/10.1016/B978-0-12-415781-1.00009-1>
- [44] Linda Upchurch, Gordon Rugg, and Barbara Kitchenham. 2001. Using card sorts to elicit web page quality attributes. *Ieee software* 18, 4 (2001), 84.
- [45] Tammy VanDeGrift, Dennis Bouvier, Tzu-Yi Chen, Gary Lewandowski, Robert McCartney, and Beth Simon. 2010. Commonsense Computing (Episode 6): Logic is Harder than Pie. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research* (Koli, Finland) (Koli Calling '10). Association for Computing Machinery, New York, NY, USA, 76–85. <https://doi.org/10.1145/1930464.1930479>
- [46] Eugene Wallingford. 1998. Elementary patterns and their role in instruction. In *OOPSLA'98*.