Learning Transfer in Novice Programmers: A Preliminary Study

Cruz Izu
The University of Adelaide
Adelaide, SA, Australia
cruz.izu@adelaide.edu.au

Claudio Mirolo
University of Udine
Udine, Italy
claudio.mirolo@uniud.it

ABSTRACT

Learning transfer refers to the ability to correctly apply learned skills, knowledge and behaviors to new situations or contexts. This paper explores novice programmers' transfer through the analysis of two related coding tasks completed by CS1 students as part of their assessment. The first task was a take-home practical and the second task was a lab practical exam; both tasks requested the implementation of a C function with an integer parameter from which the digits are to be extracted and operated on. The solution set generated from each task by a cohort of 255 CS1 students has been explored and classified in order to determine the extent of transfer from the practice task to the later assessment task.

This classification shows 36.5% of students consolidated or extended the acquired skills and 13% at least partly; 38%, on the other hand, failed to recall their previous valid strategy or to devise a better one, and were unsuccessful in the second task. On the positive side, 9% of students devised a different and improved strategy in the exam, indicating additional learning had occurred in between the two tasks. Peer review of key coding tasks could improve transfer by forcing weaker students to compare and evaluate different design strategies.

KEYWORDS

CS1, learning transfer, subgoals, code reuse, code extension

ACM Reference Format:
Cruz Izu and Claudio Mirolo. 2021. Learning Transfer in Novice Programmers: a Preliminary Study. In *26th ACM Conference on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2021), June 26-July 1, 2021, Virtual Event, Germany.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3430665.3456336

1 INTRODUCTION

This study reviews the struggles of novice programmers through the lens of Perkins' work, who covered both the acquisition of programming knowledge [20] and the conditions for transfer [22]. Perkins establishes transfer in teaching to refer to the degree at which a person is able to apply learned knowledge and skills to varying degrees of different situations and conditions.

To master programming skills, students should tackle a new problem by identifying at an early stage what previously learned knowledge or skills can be applied to that new situation. This recall will become more prevalent in later courses and in their careers as programming tasks become increasingly complicated. Sometimes the concept, such as sub-goal implementation, is learned but that knowledge is inert [24]: the student needs a cue to identify the fact that sub-goal plays a role in building the solution to a different problem. In short, transfer plays a major role on building and improving programming skills, but has received limited exploration. For example, in the recent review of research into introductory programming [16], the authors mention only one instance of work explicitly addressing learning transfer ([4]).

This preliminary study explores sub-goal transfer at CS1 level by focusing on the apparent failure to transfer a non-trivial operation (extracting the digits of an integer) from practice to exam conditions. By analysing the transfer between two similar function-writing tasks in a large CS1 cohort, we hope to characterise and comprehend the levels of transfer for novice programmers. More specifically, we will address the following research questions:

RQ1. How many students transfer their approach from practice to the exam? were they successful?

RQ2. When transfer fails, what can we learn from the students' solutions in regard to their failure?

In the long term, a better comprehension of how/when transfer occurs while learning to code could also help to identify factors that could facilitate or inhibit transfer.

2 BACKGROUND

Starting from the work by Thorndike and collaborators in the early years of 1900 (see e.g. [32]), a variety of studies have addressed *transfer* in educational contexts at different depths and in different domains, as explained in the next two sections.

2.1 Transfer definition and related work

Perkins and colleagues made relevant contributions to this area by providing deep insights [24] into defining the concept and the way in which it occurs from an educational psychology perspective, [21, 22]. They differentiate between near and far transfer.

Near transfer describes the process where well developed skills are automatically replicated or straightforwardly used when presented with a perceptually similar situation.

Far transfer occurs when the knowledge is to be applied into a context that is unlike any previous contexts where that knowledge was learned and used. Far transfer requires the use of abstraction to ignore the contextual differences and identify the core similarities.

Perkins and Salomon presented a three-step model, *detect–elect–connect*, to analyse the process of transfer in terms of "detecting a

---

ITiCSE 2021, June 26-July 1, 2021, Virtual Event, Germany

ACM ISBN 978-1-4503-8214-4/21/06…$15.00
https://doi.org/10.1145/3430665.3456336

potential relationship with prior learning, electing to pursue it, and working out a fruitful connection" [21].

Other studies analyse the different kinds of "similarity" involved in transfer processes and the role of long-term memory [9], the implications for transfer of focusing on domain-specific vs. domain-general knowledge [30], or the conditions constraining the approaches to the tasks in which transfer is usually assessed [2]. In fact, "transfer is not a static concept", as pointed out by Ford, instead

> transfer must be defined within the context of what is relevant to the type of educational intervention [6].

For example, a study looked at the transfer of debugging skills from LOGO to non-programming domains [12] and specified a debugging skill model that focuses on the importance of gathering clues to a bug's identity and its location. Then, they looked for these activities in other domains as measurement of positive transfer.

Other works have investigated the factors that could impact learning transfer in problem solving activities, including the order in which the tasks are presented to students [15] and the reciprocal interaction of group work and individual work [1]. This bulk of work, however, does not specifically address computing topics.

2.2 Transfer in Programming

Since the birth of an educational perspective on computing, in the early '80s recurrent claims about the transfer potential from learning programming to the development of higher-order thinking skills, including those required in mathematics and science [27], have been subjected to closer scrutiny. Pea and Kurland, for instance, critically reviewed previous research and beliefs [19]; Pirolli and Recker investigated the impact on transfer of a variety of teaching conditions as well as of learners' cognitive and meta-cognitive skills [23]. Later, also transfer coming from other subjects, e.g. from algebraic substitution to the understanding of recursion [14], has been occasionally explored.

Perkins and Salomon use a specific programming example to describe *inert knowledge*: when a student fails to detect and connect learned skills and knowledge if faced with new contexts [20].

While there is not yet fully convincing evidence of transfer effects of programming skills on high-level mental functions [25], more recently there have been a few attempts to take a transfer-of-learning perspective in order to pursue less ambitious objectives, such as characterising the cognitive factors affecting transfer [10], devising teaching styles and instruments tailored to facilitate transfer [8, 17, 18], relating code-writing to code-tracing skills [13], or assessing the pedagogical potential of analogy on the understanding of some specific programming concepts [4]. In addition, a little more explored area of interest concerns the extent of concept and skill transfer between different programming languages and environments, either visual/block-based versus textual [5, 7] or both textual (often pertaining to different paradigms) [26, 28, 31, 33].

2.2.1 Transfer in CS1. In CS1, most concepts are taught once and revisited or reused under a range of scenarios. An activity or coding task provides an opportunity to learn an item (for example a new concept, method or scheme). A subsequent task can be approached or resolved using that item. In short, we are frequently expecting students to achieve near transfer of coding skills from a workshop exercise to a practical one, or from a lecture example to a classroom task. When practicing code design and problem decomposition we expect students to remember sub-goals from previous tasks instead of designing from scratch.

However, it appears that transfer has not been widely explored at a finer-grain level, namely when novices engage in a series of small tasks in which they are expected to be able to apply some learned concept or (slightly) adapt some learned technique. As Teague and Lister point out, "It is a common source of frustration for computer science educators that novices do not transfer to a second programming problem the concepts taught on an initial problem" [29].

It is precisely this kind of transfer that is the subject of the present exploratory study. The focus of [29] is on the reasons of the lack of transfer, that the authors explain in terms of behaviours characteristic of the pre-operational stage within a neo-Piagetian framework. Here, on the other hand, we attempt to analyse and categorise different levels of transfer between two related small programming tasks for which both *near* and *far* transfer can be manifested. An additional difference between the two studies is that Teague and Lister's tasks focus primarily on program's operational behaviour, whereas ours also require to establish connections with an extrinsic problem domain.

3 METHODOLOGY

This project aims to build on this background by exploring transfer in a natural context where the data has been sourced from activities within a normal course instead of a controlled experiment. It also intends to view the occurrence of transfer in the current landscape as computer science continues to change and evolve.

3.1 Data Collection

The data collected is comprised of solutions to two similar coding tasks by students taking our CS1 introductory course in 2015. Fig. 1 shows the two problems that students were asked to code a solution for. For the formative task, students were to explain their code in person to the course tutors in order to get their mark. This verbal explanation played multiple roles in their learning: it discourages cheating by having to understand any code they source online; it helps to improve their technical skills in presenting code and having an opportunity for one-to-one feedback. In practice, large classes with a tutor to students' ratio of 1/15 added pressure to both mark and help students in the same session, hence reducing these benefits.

Both functions receive a number as a parameter, and their solutions share a common subgoal: extract the digits from the integer; a second subgoal involves checking a condition related to those digits. As the conditions to be checked do not overlap between the two tasks, we focus on the transfer for the first sub-goal.

3.2 Data Analysis

The goal of this analysis is to identify whether transfer has occurred from the formative task (week 10, *isArmstrong*) to the summative task (week12, *isFortunate*).

A previous study [11] analysed the *isFortunate* exam solutions and found that several students were challenged by the digit extraction in spite of the fact they had implemented 3-digit extraction

An Armstrong number (AN) of three digits is an integer such that the sum of the cubes of its digits equals the number itself.
For example, 407 is an Armstrong number since

$$4^3 + 0^3 + 7^3 = 407$$

Write a function called **isArmstrong** that receives an integer in the range 100-999. The function returns 1 if the number is an AN, otherwise it returns 0.

(a) Formative task - Homework exercise

We have decided that 3 and 7 are fortunate digits. A positive integer is called a *fortunate* number if its decimal representation has only fortunate digits, i.e., 3, 37, 737.

Write a function in C called **isFortunate** that receives an integer number as a parameter, and returns

- 1 if the number if fortunate,
- 0 if is the number is not fortunate
- -1 if the number is negative.

(b) Summative task - Practical exam question

Figure 1: Task under analysis for transfer from formative to summative assessment

in the formative task. The main difference between the two tasks is the range of the `int` parameter: in the homework exercise it is delimited by the range 100–999, while the values in the practical exam have no predefined range. This means students may need to generalise their approach to extract digits from a fixed-length number to work with numbers of variable length.

Our working hypothesis is that low transfer was due to a failure to generalise. Thus, we will focus our analysis on the first subgoal for both functions in order to explore transfer between the tasks. The task's analysis was carried out in three phases: *Phase 1*: Identify and classify the methods used by students to extract the digits in *isArmstrong*. *Phase 2*: Identify and classify the methods used by students to extract the digits in *isFortunate*. *Phase 3*: determine the transfer that occur from formative to summative task.

In the first two phases, each student's file was examined and tagged by two of the authors. Both coders agreed on the key methods but used at times different names for the minor variations. In regard to phase 3, we expected to code transfer as full, partial or non-existent. However, after one coding iteration and some discussion we identified four possible transitions:

Extended transfer This is a positive transfer event, in which the previous implementation of the sub-goal is extended beyond basic transfer.

Consolidation The sub-goal code used in the context of the prior task is recalled and reused in the new context. This is a positive transfer event that will consolidate the acquired coding technique.

Partial transfer The sub-goal code used in the prior task is recalled and/or used with some error or omission.

Failed transfer A poor attempt to recall the code used in the prior task.

In that iterative process, we also identified two different non-transfer transitions:

No transfer Instead of using the previous approach, the sub-goal is not implemented or a different and inadequate approach is selected. Such action indicates the previous method was not learnt as it could not be recalled.

New insight a new and suitable strategy was used as an alternative to implement the sub-goal. In this case we cannot conclude the previous approach was not learnt, as it may have been purposely replaced with the new one.

Note when plotting the results, we have collated "extended transfer" and "new insight" together under "Improved".

4 RESULTS

4.1 isArmstrong implementation

An analysis of the solutions submitted for the Armstrong Numbers question was performed to first categorise the methods used by students to extract digits. As shown in the code examples in Fig. 2, students used either division or modulo to extract each digit. In order to shift to the next value, students used a variety of operations including combinations of multiplication and subtraction, subtraction and division or just division. The analysis identified three extraction methods:

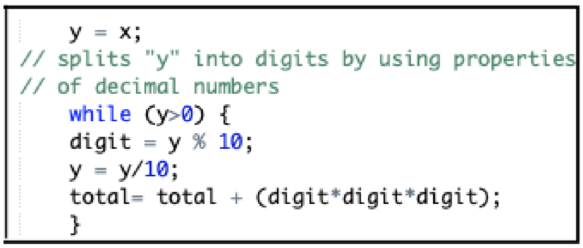
While iteration: a while loop that extracts the lowest digit and removes it until the top digit is removed. This method, shown in Fig. 2.(a), is the general method to extract digits from a number.

For iteration: a for loop that iterates three times, each iteration extracting and operating on one digit, as shown in Fig. 2.(b).

Hard-coded command sequence: having a statement for each digit that extract its value using a mathematical computation. The extraction can start from the top digit, as in Fig. 2.(c), or from the lowest digit, Fig. 2.(d).

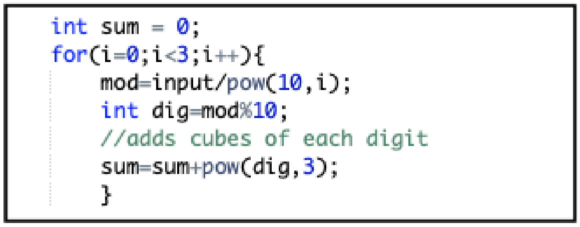
As novice programmers, students are likely to choose a simple method from their coding perspective. The nature of the isArmstrong description meant that students were likely to hard-code the solution as the input was always length 3. Hence, it is not surprising that 56% of the cohort used that method, followed by 20.3% that used the more general *while* method, 11.3% iterated on a *for* loop and 10% used other methods.

Fig. 3 shows the distribution of the methods listed above for the two tasks. In the formative task, 56% of students hard-coded the digit extraction (70% of them chose to extract the top digit first), followed by 27% using the efficient and general *while* loop; 11% of students iterated with a *for* loop, while 6% (15 submissions) use other approaches that skip the need to extract digits. Of those 15 solutions, 4 were valid: three students used exhaustive enumeration, a correct approach with limited reuse, as it relies of finding or generating the list of solutions; one student casted the integer into a string by calling the *sprintf* method; invalid approaches skipped



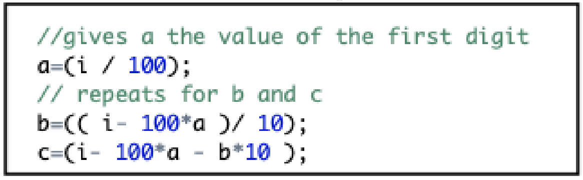
```
    y = x;
// splits "y" into digits by using properties
// of decimal numbers
    while (y>0) {
    digit = y % 10;
    y = y/10;
    total= total + (digit*digit*digit);
    }
```

(a) While loop



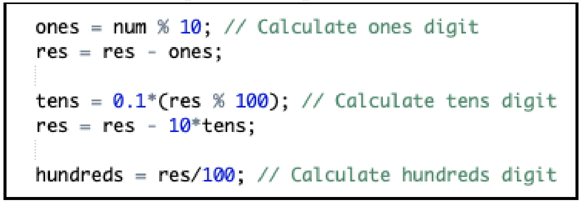
```
  int sum = 0;
  for(i=0;i<3;i++){
      mod=input/pow(10,i);
      int dig=mod%10;
      //adds cubes of each digit
      sum=sum+pow(dig,3);
      }
```

(b) For loop



```
//gives a the value of the first digit
a=(i / 100);
// repeats for b and c
b=(( i- 100*a )/ 10);
c=(i- 100*a - b*10 );
```

(c) A top-first example of hardcoded



```
ones = num % 10; // Calculate ones digit
res = res - ones;

tens = 0.1*(res % 100); // Calculate tens digit
res = res - 10*tens;

hundreds = res/100; // Calculate hundreds digit
```

(d) A bottom first example of hardcoded

Figure 2: Digit extraction strategies for *isArmstrong*.

extraction by incorrectly changing the input type to be string based or by asking the user to enter one digit at a time.

We should note that when the exam task was set, the teacher's expectation was an iterative approach will be widely used, so that the exam task is solved using near transfer. As only 38% of students applied an iterative approach, this means the exam task became for most students a challenge in generalization to make their hard-coded method work with different digit lengths.

4.2 isFortunate implementation

The extraction of digits from the integer parameter differed from the previous tasks in that we cannot make any assumption on its number of digits. Thus, any correct and complete digit extraction approach needs to be adapted to extract exactly the number of digits present for a given input. This is already supported by the "while" approach to the previous task. It can be coded as an extension to the "for" and "hard-code" methods. An example of such extension, restricted to numbers in the [0,999] range is shown in Fig 4.

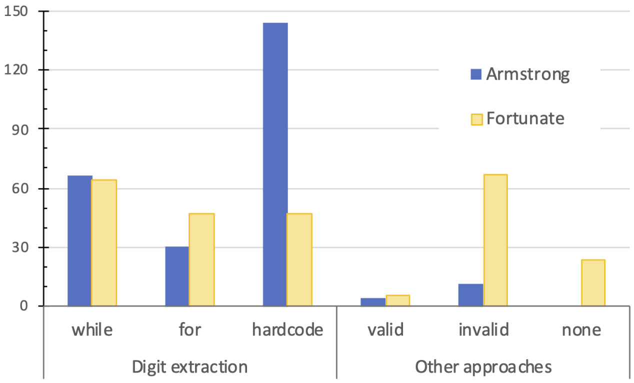
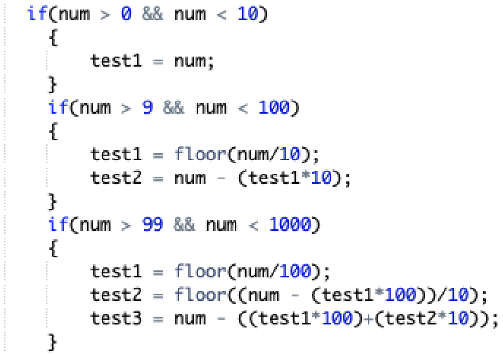


Figure 3: Digit processing strategies distribution for each coding task (n=255).



```
if(num > 0 && num < 10)
  {
      test1 = num;
  }
  if(num > 9 && num < 100)
  {
      test1 = floor(num/10);
      test2 = num - (test1*10);
  }
  if(num > 99 && num < 1000)
  {
      test1 = floor(num/100);
      test2 = floor((num - (test1*100))/10);
      test3 = num - ((test1*100)+(test2*10));
  }
```

Figure 4: An example of applying the hard-code approach to *isFortunate* by testing parameter range.

This type of partial extension was used by 24 students, compared to 15 students that just reused their previous hardcode, which will add one or two leading zeros numbers in the range [0,99]. Students reusing a *For* loop approach needed to extend it by determining the number of digits for the given input. They use a while loop preceding the *for* loop that counts the number of times the number can be divided by 10 until it becomes 0 (or the number of times I can multiply a variable set to 1 until is greater than the number).

As shown in Fig. 3, exam conditions resulted in lower performance for that sub-goal with 67 students using invalid approaches. The set of invalid approaches includes incomplete enumeration (10 students), changing the parameter to string to avoid conversion issues (22), operating the integer value as having an additional string (7) or integer array (17), and poor coding approaches with no clear strategy (11).

4.3 Transfer from isArmstorng to isFortunate

Next we investigated whether students were reusing the same sub-goal approach (digit-extraction) in the second task. It will seem reasonable to expect learning transfer to correlate somehow with previous performance. Hence, we will analyse these patterns of transfer in detail by splitting the student cohort into three different subsets relative to their preparation for the exam task: (1) well-prepared, because they already used an iterative approach to extract the digits, so they had a viable approach to reuse; (2) somehow-prepared, as they know how to extract individual digits or used another viable approach; (3) poorly-prepared, as they used other, mostly incorrect approaches for the formative task.

Table 1 provides a summary of the results, while Figs. 5 and 6 provide further details for the two main subsets.

4.3.1 Transfer from iterative approach. Students that applied an iterative approach (n=96) were, on paper, in a good position to reuse their approach in the exam task. However, as Fig. 5 shows, only 46% of those students were able to reuse or extend their approach two weeks later. Learning transfer was worse for the students that used the simpler *while* loop: 10% making no attempt and 41% choosing invalid strategies and coding them poorly.

In comparison, the results for the students that previously used a *for* loop are more encouraging: only 26% failed to recall their approach, and 4 students from that cohort where able to extend that approach by using the while loop, hence moving beyond consolidation towards extended transfer.

4.3.2 Transfer from hard-coded approach. The performance of this large set (n=144) is overall quite positive considering they were handicapped in terms of having to extend their previous code to cope with variable digit length.

As expected under exam conditions, nearly 30% could not recall their previous approach: 20% of students chose poor strategies such as changing the input type to string to facilitate extraction, and 8% provided a nearly empty solution. Finally, 2 students used enumeration to validate any integers with ≤ 3 digits, and 1 student cast the integer into a string and then iterated over the digits.

The level of recall or transfer is 63.9% (improve, consolidate or achieve partial transfer) as shown in Fig. 6. We speculate that most students in this group wrote their own code for the formative task after playing with 3-digit values on paper. This assumption matches the myriad of slightly different solutions. Although their designs were not elegant or efficient, their hands-on approach supported students' learning of how to manipulate integers at the digit level; thus, many were able to recall and adapt their approach two weeks later. In fact, 40% of them attempted an iterative approach, and 33 students (23% of them) showed significant improvement beyond near transfer: 14 of them achieved extended transfer, usually using a *for* loop that divides the number by powers of 10. The remaining 19 students used a simple an efficient while loop, which we classified as *new insight* as they have been optimised. Some of those students may have discussed their solutions with peers that used iterative solutions and learned from them to improve their approach.

4.3.3 Transfer from other approaches. This is the smallest subset (n=15). Most students chose poor strategies in the formative task. Using the same bad strategy is not a sign of transfer but of stagnation and this was the case for 4 of them. On the positive side, 5 chose a better approach (either *for* or *hardcode*) and 2 of them were successful in their implementation.

Three out of the 4 students that succeed in the formative task consolidated that knowledge by adapting it to the exam task: one cast the integer into a string in order to extract the digits, and two students used enumeration of fortunate numbers with length ≤ 3.

Table 1: Transfer from formative to summative task

	isArmstrong approach			
Transfer	**Iterative**	**Hardcode**	**Other**	*Totals*
New Insight	0	19	3	22 (8.6%)
Extended	4	14	0	18 (7%)
Consolidate	37	35	3	75 (29%)
Partial	9	24	1	34 (13%)
Failed	4	6	6	16 (6%)
No transfer	42	46	2	90 (35%)
Total	96	144	15	255

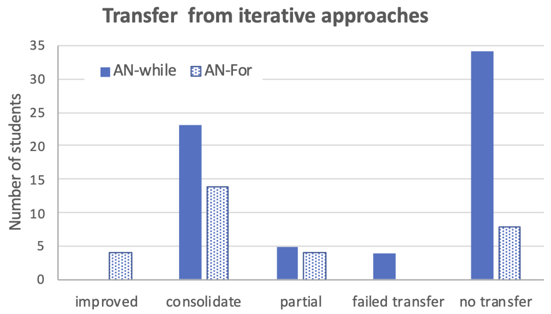


Figure 5: Evaluation of transfer for students that initially used "while" or "for" methods for digit extraction (n=96)

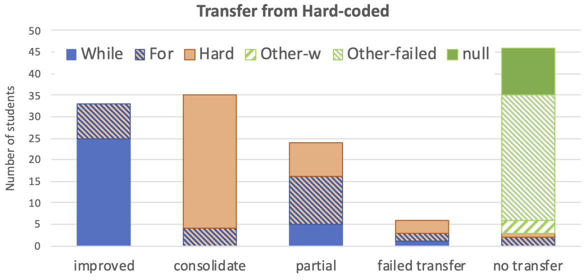


Figure 6: Evaluation of transfer for students that initially used hardcode methods for digit extraction (n=144)

5 DISCUSSION

The comparison of strategies used in practice versus exam to achieve the sub-goal (see Fig. 3) provides a typical picture of lower performance under exam conditions: 11% made no attempt and 25% could not recall their previous valid approach to complete that sub-goal. Next, we will consider the learning transfer by measuring their success relative to their earlier attempt to extract digits.

5.1 Research findings

RQ1 - How many students transfer their approach from practice to the exam? were they successful? 143 students (56%) tried to reuse their digit extraction approach in the exam, but only 36.5% of students were successful. 13% of all students achieved partial transfer and 6% of them failed to transfer. On the positive side, 9% of students devised a different or improved strategy in the exam, indicating additional learning had occurred in between the two tasks. Overall, this represents a relative success rate of 58.4%, which is considered a good outcome for this extended coding task.

RQ2 - When transfer fails, what can we learn from the students' solutions in regard to their failure? We have identified two patterns of failing transfers that reflect a significant lack of coding skills:

- **No recall** : (28 students - 11%) they made no attempt to implement any sub-goals. This may be due to minimal recall or lack of code fluency.
- **Poor attempts** with meaningless or complicated code (41 students - 16%): their strategy is unclear or uses a variable as being an integer and a char/int array at the same time.

We may envisage different explanations for lack of transfer. Some of these students may simply belong in the *"Following"* category, according to the phenomenographic taxonomy in [3], experiencing learning to program as merely getting through the assignments. On the other hand, more interestingly, others may get stuck because of the interactions between the transfer process and the mechanisms of *surface* and *structural* similarity, pinpointed by Gentner et al. [9].

From Table 1 we can see that 46 students, i.e., 18%, were unable (or failed) to transfer to the *isFortunate* context the almost ready scheme of their iterative solution to the formative task. In fact, 37% of poor attempts were made by students from that subset: 5 changed the input to string, 11 treated an integer as a digit string, 3 used partial enumeration and 8 were unclear in their approach. It is likely these students sought help (online or from peers) to complete the formative task, with limited effort to learn from it. Partial transfer (13%) refers to students that have gained some problem solving and coding skills along the way but need more practice in order to correct their mistakes.

In short, this study has measured learning transfer to range from 36% (consolidate or extended transfer) to 59% (when including partial transfer and new insights). Additionally, the analysis of learning transfer has provided the following insights: (1) most of the learning improvements come from students that used a hard-coded approach in the first task and (2) the students using the more efficient *while* approach exhibit the lowest transfer.[1]

5.2 Threats to validity

One limitation of this study is that the time constrains and stress in the practical exam may have reduced the transfer. On the other hand, this task was worth 2.5% of their final mark so it would have motivated students to revise their prior work. A second limitation is that the first task was a take-home exercise which provides opportunities to consult and learn from other sources, which is a regular scenario for most CS1 tasks. An experiment in which the first task is completed in a controlled environment will avoid this drawback. On the other hand, most of the learning for novice programmers occurs on their home practice, so the current experiment is adequate for measuring transfer under normal conditions.

Although online searching is a potential first step toward building the solution, students were made aware of the need to use this for design ideas instead of merely copying and pasting. Having to explain the submitted code to a tutor to get the marks was, in hindsight, a limited strategy that required students to be self-motivated to grasp the coded concepts instead of just describing them.

This is a preliminary study with a single student cohort and one sub-goal transfer task, so further studies are needed to cover multiple cohorts and multiple tasks in order to validate the model proposed to measure the level of transfer for CS1 sub-goals.

5.3 Implications for educators

Near transfer describes the process where a learned skill under a given context is activated and used in a similar context. In CS1, most concepts are taught once and revisited and reused in similar and different exercises. Intuitively, most educators rely on transfer for CS1 students to build their coding experience by reusing and combining sub-goals they have implemented before when solving larger problems.

This study confirms the impact of formative assessment and student's motivation into their exam performance. Performance, both in marks and in code quality on the practical task plays only a part on students' learning. We hypothesize that the extended transfer observed in 30 students from the hard-coded subgroup is the result of feedback and *ad-hoc* code review with their peers. Motivated students learn from their practice and consolidate and extend their knowledge. Unmotivated students may perform well in homework tasks while learning little from them due to lack of reflection and comprehension.

6 CONCLUSIONS AND FURTHER WORK

Learning transfer occurs at multiple levels: syntax, code structure, commands, algorithms. When focusing on code design, transfer facilitates code reuse, which is critical to build computational thinking and problem solving skills. Although we expect CS1 students to transfer their new skills into later tasks, little is known in regard to when transfer occurs and which factors we should consider in our course delivery and organization to improve transfer.

In this study we have focused on the transfer of sub-goals. Using the data from a large CS1 cohort, we have proposed a model to measure the level of transfer between two related tasks for a concrete CS1 sub-goal: digit extraction. The analysis provides not only a description of the different strategies used by students, but also some clues on the depth of their learning.

This is a preliminary study and more work is required to test this model in different scenarios. Future work should also explore factors, particularly those linked to instruction, that facilitate learning transfer of sub-goals. On that point, it could be interesting to both consider the "transfer-oriented instruction" guidelines suggested in [10] and to explore the potential of peer collaborations, as discussed in [34], in order to foster learning transfer through reflection on their own strategies.

---

[1] The correlation of better patterns of transfer to "hard-coded" solutions in the first task appears statistically significant via χ^2-test on a contingency table ($p < 10^{-3}$).

REFERENCES

[1] Patrice Belleville, Steven A. Wolfman, Susanne Bradley, and Cinda Heeren. 2020. Inverted Two-Stage Exams for Prospective Learning: Using an Initial Group Stage to Incentivize Anticipation of Transfer *(SIGCSE '20)*. Association for Computing Machinery, New York, NY, USA, 720–738. https://doi.org/10.1145/3328778.3366938

[2] John D. Bransford and Daniel L. Schwartz. 1999. Rethinking Transfer: A Simple Proposal With Multiple Implications. *Review of Research in Education* 24, 1 (1999), 61–100. https://doi.org/10.3102/0091732X024001061

[3] Christine Bruce, Lawrence Buckingham, John Hynd, Camille Mcmahon, Mike Roggenkamp, and Ian Stoodley. 2004. Ways of experiencing the act of learning to program: A phenomenographic study of introductory programming students at university. *Journal of Information Technology Education* 3 (2004), 143–160.

[4] Yingjun Cao, Leo Porter, and Daniel Zingaro. 2016. Examining the Value of Analogies in Introductory Computing. In *Proceedings of the 2016 ACM Conference on International Computing Education Research* (Melbourne, VIC, Australia) *(ICER '16)*. Association for Computing Machinery, New York, NY, USA, 231–239. https://doi.org/10.1145/2960310.2960313

[5] Wanda Dann, Dennis Cosgrove, Don Slater, Dave Culyba, and Steve Cooper. 2012. Mediated Transfer: Alice 3 to Java. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education* (Raleigh, North Carolina, USA) *(SIGCSE '12)*. Association for Computing Machinery, New York, NY, USA, 141–146. https://doi.org/10.1145/2157136.2157180

[6] J. Kevin Ford. 1994. Defining Transfer of Learning: The Meaning Is in the Answers. *Adult Learning* 5 (March 1994), 22–30. https://doi.org/10.1177/104515959400500412

[7] Diana Franklin, Charlotte Hill, Hilary A. Dwyer, Alexandria K. Hansen, Ashley Iveland, and Danielle B. Harlow. 2016. Initialization in Scratch: Seeking Knowledge Transfer. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (Memphis, Tennessee, USA) *(SIGCSE '16)*. Association for Computing Machinery, New York, NY, USA, 217–222. https://doi.org/10.1145/2839509.2844569

[8] S. García-Martínez and Daniel Zingaro. 2011. Teaching for Transfer of Learning in Computer Science Education. *ISTE Journal for Computing Teachers (JCT)* (2011).

[9] D. Gentner, M.J. Rattermann, and K.D. Forbus. 1993. The Roles of Similarity in Transfer: Separating Retrievability From Inferential Soundness. *Cognitive Psychology* 25, 4 (1993), 524–575. https://doi.org/10.1006/cogp.1993.1013

[10] David Ginat, Eyal Shifroni, and Eti Menashe. 2011. Transfer, Cognitive Load, and Program Design Difficulties. In *Informatics in Schools. Contributing to 21st Century Education*, Ivan Kalaš and Roland T. Mittermeir (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 165–176. https://doi.org/10.1007/978-3-642-24722-4_15

[11] C. Izu and P. Dinh. 2018. Can Novice Programmers Write C Functions?. In *2018 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*. 965–970.

[12] David Klahr and Sharon M. Carver. 1988. Cognitive objectives in a LOGO debugging curriculum: Instruction, learning, and transfer. *Cognitive Psychology* 20, 3 (1988), 362–404. https://doi.org/10.1016/0010-0285(88)90004-7

[13] Amruth N. Kumar. 2015. Solving Code-Tracing Problems and Its Effect on Code-Writing Skills Pertaining to Program Semantics. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education* (Vilnius, Lithuania) *(ITiCSE '15)*. Association for Computing Machinery, New York, NY, USA, 314–319. https://doi.org/10.1145/2729094.2742587

[14] Colleen M. Lewis. 2014. Exploring Variation in Students' Correct Traces of Linear Recursion. In *Proceedings of the Tenth Annual Conference on International Computing Education Research* (Glasgow, Scotland, United Kingdom) *(ICER '14)*. Association for Computing Machinery, New York, NY, USA, 67–74. https://doi.org/10.1145/2632320.2632355

[15] Nan Li, William W. Cohen, and Kenneth R. Koedinger. 2012. Problem Order Implications for Learning Transfer. In *Intelligent Tutoring Systems*, Stefano A. Cerri, William J. Clancey, Giorgos Papadourakis, and Kitty Panourgia (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 185–194.

[16] Andrew Luxton-Reilly, Simon, Ibrahim Albluwi, Brett A. Becker, Michail Giannakos, Amruth N. Kumar, Linda Ott, James Paterson, Michael James Scott, Judy Sheard, and Claudia Szabo. 2018. Introductory Programming: A Systematic Literature Review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education* (Larnaca, Cyprus) *(ITiCSE 2018 Companion)*. ACM, New York, NY, USA, 55–106. https://doi.org/10.1145/3293881.3295779

[17] Lauren E. Margulieux, Mark Guzdial, and Richard Catrambone. 2012. Subgoal-Labeled Instructional Material Improves Performance and Transfer in Learning to Develop Mobile Applications. (2012), 71–78. https://doi.org/10.1145/2361276.2361291

[18] Briana B. Morrison, Lauren E. Margulieux, and Mark Guzdial. 2015. Subgoals, Context, and Worked Examples in Learning Computing Problem Solving. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research* (Omaha, Nebraska, USA) *(ICER '15)*. Association for Computing Machinery, New York, NY, USA, 21–29. https://doi.org/10.1145/2787622.2787733

[19] Roy D. Pea and D. Midian Kurland. 1984. On the cognitive effects of learning computer programming. *New Ideas in Psychology* 2, 2 (1984), 137– 168. https://doi.org/10.1016/0732-118X(84)90018-7

[20] David Perkins, Fay Martin, and Cambridge, MA. Educational Technology Center. 1985. *Fragile Knowledge and Neglected Strategies in Novice Programmers. IR85-22 [microform] / David Perkins and Fay Martin.* Distributed by ERIC Clearinghouse [Washington, D.C.]. 35 p. pages. http://www.eric.ed.gov/contentdelivery/servlet/ERICServlet?accno=ED295618

[21] David N. Perkins and Gavriel Salomon. 2012. Knowledge to Go: A Motivational and Dispositional View of Transfer. *Educational Psychologist* 47, 3 (2012), 248–258. https://doi.org/10.1080/00461520.2012.693354

[22] David N. Perkins, Gavriel Salomon, and Pergamon Press. 1992. Transfer Of Learning. In *International Encyclopedia of Education (2nd.* Pergamon Press.

[23] Peter Pirolli and Margaret Recker. 1994. Learning Strategies and Transfer in the Domain of Programming. *Cognition and Instruction* 12, 3 (1994), 235–275. https://doi.org/10.1207/s1532690xci1203_2

[24] Gavriel Salomon and David N. Perkins. 1989. Rocky Roads to Transfer: Rethinking Mechanism of a Neglected Phenomenon. *Educational Psychologist* 24, 2 (1989), 113–142. https://doi.org/10.1207/s15326985ep2402_1

[25] Ronny Scherer. 2016. Learning from the Past?The Need for Empirical Evidence on the Transfer Effects of Computer Programming Skills. *Frontiers in Psychology* 7 (2016), 1390. https://doi.org/10.3389/fpsyg.2016.01390

[26] J. Scholtz and S. Wiedenbeck. 1992. An analysis of novice programmers learning a second language. In *PPIG*.

[27] Sylvia A. Shafto. 1986. Programming for Learning in Mathematics and Science. In *Proceedings of the Seventeenth SIGCSE Technical Symposium on Computer Science Education* (Cincinnati, Ohio, USA) *(SIGCSE '86)*. Association for Computing Machinery, New York, NY, USA, 296–302. https://doi.org/10.1145/5600.5635

[28] N. Shrestha, T. Barik, and C. Parnin. 2018. It's Like Python But: Towards Supporting Transfer of Programming Language Knowledge. In *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 177–185. https://doi.org/10.1109/VLHCC.2018.8506508

[29] Donna Teague and Raymond Lister. 2014. Manifestations of Preoperational Reasoning on Similar Programming Tasks. In *Proceedings of the Sixteenth Australasian Computing Education Conference - Volume 148* (Auckland, New Zealand) *(ACE '14)*. Australian Computer Society, Inc., AUS, 65–74.

[30] André Tricot and John Sweller. 2014. Domain-Specific Knowledge and Why Teaching Generic Skills Does Not Work. *Educational Psychology Review* 26, 2 (2014), 265–283. https://doi.org/10.1007/s10648-013-9243-1

[31] Ethel Tshukudu and Quintin Cutts. 2020. Understanding Conceptual Transfer for Students Learning New Programming Languages. In *Proceedings of the 2020 ACM Conference on International Computing Education Research* (Virtual Event, New Zealand) *(ICER '20)*. Association for Computing Machinery, New York, NY, USA, 227–237. https://doi.org/10.1145/3372782.3406270

[32] R. S. Woodworth and E. L. Thorndike. 1901. The influence of improvement in one mental function upon the efficiency of other functions. *Psychological Review* 8, 3 (1901), 247–261. https://doi.org/10.1037/h0074898

[33] Quanfeng Wu and John R. Anderson. 1990. *Problem-solving transfer among programming languages.* Technical Report AIP-134. Carnegie-Mellon University, Pittsburgh, PA. The Artificial Intelligence and Psychology Project.

[34] Daniel Zingaro and Leo Porter. 2015. Tracking Student Learning from Class to Exam Using Isomorphic Questions. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (Kansas City, Missouri, USA) *(SIGCSE '15)*. Association for Computing Machinery, New York, NY, USA, 356–361. https://doi.org/10.1145/2676723.2677239