7th International Conference on Higher Education Advances (HEAd'21)
Universitat Politècnica de València, València, 2021
DOI: http://dx.doi.org/10.4995/HEAd21.2021.13115

Analogy-based Instruction for Effective Teaching of Abstract Concepts in Computer Science

Pawan Saxena[1], Sanjay K. Singh[1], Gopal Gupta[2]

[1]Department of Information Technology, Amity University, Lucknow, India, [2]Department of Computer Science, University of Texas at Dallas, USA.

Abstract

In the analogy-based learning method we map a concept that is being learned to a well-understood concept. An analogy is mainly useful when learners lack prior knowledge of the topic being learned. Computer Science (CS) is a subject whose concepts tend to be highly abstract and therefore difficult for undergraduate students to understand. Analogy-based instruction can greatly reduce a student's burden of learning these abstract CS concepts. Role of analogy in teaching CS topics has not been adequately explored. In this paper we discuss analogy-based instruction in computer science and its advantages. Over the last decade we have developed analogies for a large number of difficult CS concepts and extensively used them in the classroom at our institution. We list these analogies and as an illustration discuss one of them (from the subfield of operating systems) in detail. We also present the evaluation of our analogy-based instruction method. Our results indicate that our techniques are quite effective in improving student learning outcomes.

Keywords: *Analogy-based instruction; CS; improving learning outcomes.*

1. Introduction

Teaching Computer Science (CS) concepts effectively to undergraduate students is a challenging task. Students find CS concepts hard to understand for two primary reasons: (i) most students are not exposed to CS concepts during their pre-university education. Even if they learn to program in high school, they will not be exposed to concepts such as how an operating system works; (ii) most CS concepts are quite abstract, as almost everything is based on interpretation of data expressed in bits and bytes; students are not used to this type of abstract representations and thinking. In this paper we outline our efforts in communicating abstract CS concepts to undergraduate students in an effective manner based on use of analogies. We have employed analogy-based instruction in the classroom now for more than a decade for teaching a large number of core CS concepts. Analogy based teaching maps abstract CS concepts (target concept) to concrete everyday concepts known to students (base concept), resulting in better understanding of the target concepts.

Analogies play a major role in "problem solving, decision making, argumentation, perception, generalization, creativity, invention, emotion, prediction, explanation, conceptualization and communication" (Wikipedia Contributors 2021). Our focus is on using analogies for effectively communicating information pertaining to CS concepts. Analogy-based teaching has been proposed for communicating complex concepts in the past (Gentner 1983, Gentner & Colhoun 2010, Hofstadter 2001). Our analogies for CS concepts follow the tenets of structure mapping theory (Gentner 1983). The structure mapping theory describes psychological processes involved in learning from analogies. The theory describes how familiar knowledge about a base domain can be used to understand a less familiar or an unfamiliar idea in the target domain. A domain consists of objects, their properties, relationship and interaction between the objects, and properties of those interactions. Analogical reasoning then involves recognizing similar structure between the target and the base domains. The closer the correspondence between the domains, the stronger the analogy. For example, one can understand the flow of electrical current by mapping it to flow of water where electric current corresponds to water, pressure differential (that causes water flow) to potential difference (that causes electric current flow), narrowing of a pipe to increased resistance, etc.

With the structure alignment theory in mind, we have designed a large number of analogies for advanced CS subject in the last 10+ years. We have observed that these analogies resulted in improved student learning outcomes. In particular, we have developed detailed analogies for these nine key advanced topics in CS: (i) Pipelining and Parallel Processing in Advanced Computer Architecture (ii) Process states in Operating Systems; (iii) Virtual Memory in OS; (iv) Chomsky's Hierarchy in Automata Theory; (v) Asymptotic Notations in Design and Analysis of Algorithms; (vi) Lists, list processing and stacks in data structures (vii) almost all topics of Software engineering; (viii) Design of CPU, Control Unit, and Cache in

Computer Architecture; (ix) Macros, Compilers, OS in Systems Programming. In this paper, we describe one of the analogies in detail. Analogy based instruction leads to better learning outcomes as students now learn by building upon their existing knowledge of everyday life. Our own extensive experience in the classroom bears this out.

2. Analogy-based Instruction in CS

Teaching CS concepts to undergraduate students is a difficult task. A major problem is that most students are unfamiliar with these concepts as they have not been exposed to them prior to attending a University. Even if they have taken programming courses in high school, they would have not been exposed to CS concepts such as those related to operating systems, compilers, computer architecture, etc. Most of the students encounter these concepts for the first time when they take CS courses as part of their undergraduate curriculum. Good examples of such hard-to-understand concepts are the notion of process and process scheduling in operating systems, cache memory and memory hierarchy in computer architecture, and the Chomsky hierarchy in automata theory. Most CS concepts tend to be rather abstract in nature, making them even harder for students to understand when they encounter them for the first time. A good pedagogical technique is to *teach students by building upon what they already know*. Due to abstract nature of CS concepts, teaching them by building upon what students already know becomes quite challenging, as their knowledge of advanced CS is minimal.

Research indicates that illustrating abstract concepts through concrete representations and drawing direct connections between them helps students understand abstract concepts better. If purely abstract terms are used, then students have difficulty in fully grasping the concept. At the same time, teaching a new concept in an exclusively concrete manner limits a student's ability to generalize and apply the concept in other contexts. In CS, most of the concepts are themselves quite abstract to begin with as they are represented using bits and bytes (for example, the concept of a process in operating systems). There is really no concrete version of these concepts. Concretization of CS concepts, hence, can *only be achieved by resorting to analogies*. Thus, the difficulties in communicating CS concepts to undergraduate CS students can be mitigated by employing an approach based on using analogies. These analogies employ everyday concepts that students are already familiar with and that they can easily relate to. The benefit of using analogies is well established (Gentner 1983, Hofstadter 2001, Gentner and Calhoun 2010). Use of analogies for teaching CS concepts has two benefits: (i) Abstract concepts become concretized, making it easier for students to understand them. Since in CS, even concrete concepts are quite abstract, resorting to analogies that map these abstract concepts to concrete concepts in other areas greatly helps in achieving better learning outcomes. Research indicates that understanding abstract concepts directly is significantly harder (Newby & Stepic 1987). (ii) Analogy based teaching reinforces the principle of pedagogy that says that a new concept should be taught by building upon the foundation of existing knowledge that the student possesses. The abstract concept's analogy to an everyday concrete concept indeed allows students to understand new concepts more easily by building upon existing knowledge.

3. Related Work

Analogical reasoning plays an outsized role in everyday life: in teaching, communication, and in research (Bertha 2019, Wikipedia contributors 2021). In the classroom, analogical reasoning can be a catalyst for achieving excellent learning outcomes, yet it has not been widely employed in CS education. Gentner (Gentner 1983) has developed the structure mapping theory to describe the processes involved in analogical reasoning. Our analogies are based on this structural mapping theory. Glynn (Glynn 1994) has developed a theory of teaching with analogies and developed six-step process of teaching with this method. The analogies we have designed for CS concepts also conform to Glynn's methodology. We have made elaborate efforts to ensure that our analogies works for minutest of details and do not break down. Gadgil and Nokes (Gadgil and Nokes 2009) showed that analogy supports collaborative learning, especially when conceptual understanding is essential. Ruef (Ruef 2011) developed a method of teaching that uses analogies in the classroom to explain topics. In this methodology, one always considers the question, "what does this topic remind you of?" The program is designed to build critical thinking skills through analogies. This method is applicable to simpler topics, perhaps at the grade school and middle school level, and hard to use for CS topics.

We next illustrate a detailed analogy for the concept of process scheduling in operating systems. Here we attempt to explain to the students how processes are scheduled on a CPU. The scheduling process is quite complex, as it involves the secondary memory (disk), the main memory, and the CPU. There is a short-term scheduler and a long-term scheduler, both of which work in tandem to produce optimal results. The concept of a process itself is quite abstract for beginning students to understand, and their difficulty in understanding is further compounded by having to deal with the scheduling of these processes for execution on the CPU. Details of process scheduling can be found in any Operating Systems textbook.

4. Concept of Processes and Process Scheduling

First, we explain the concept of process scheduling in Operating Systems. A program in secondary memory is in a sleeping state, until activated by the user or by the system when it changes to a process. At a given moment all the activated programs (which are now processes) are scheduled by the long-term scheduler which forms a job queue. Some processes of the job queue are scheduled by long-term scheduler to be residing in main memory and this results into a ready queue (they wait in ready queue for their turn, they are scheduled by short-term scheduler and CPU is allocated to them). As processes are generated, a PCB (Process Control Block) is created for them. A PCB has (a) process state (b) Program counter (c) CPU registers (d) CPU scheduling information (e) Memory management information (f) Accounting Information (g) I/O status information.

A short term scheduler picks one of the processes from the ready queue and allocates the CPU to it. Which process will be selected is based on the job scheduling algorithm (FIFO, Round robin, Priority selection, Shortest job first). This process is executed by CPU till either this process is interrupted by OS, I/O occurs for the process, or the process terminates. Process may carry out I/O bound activities and its state, flag register and other register values are saved in PCB. Next process is scheduled by short-term scheduler and CPU starts executing this process. This process may either be I/O bound or is fully executed by CPU and terminates. Once the I/O for the process finishes, process comes back in the ready queue and is once again in contention for CPU. The processes which are interrupted by the OS are also waiting in the ready queue. Some processes are partially executed, (due to long duration of I/O) they are swapped out of the RAM to the hard disk and are out of active contention of the CPU (no more in ready queue). These processes are scheduled for a later time when they will be again brought into memory; their execution continues from where it was interrupted. Scheduling of these processes is performed by medium term scheduler.

Switching the CPU to another process requires saving the state of old process and loading the state for the new process. This task is known as context switching. The context of a process is available in the PCB of a process, it includes the value of the CPU registers and the process state. When a context switch occurs, the kernel saves the context of the old process in PCB and loads the saved context of the new process scheduled to run. Context switching is a pure overhead because the CPU does not do any useful work while switching. If a process after moving through various states is over, it exits the system and is available in the hard disk with some new generated results.

We next give an analogy to CPU process scheduling using the example of a hospital. There is a close (one-one) mapping between various attributes of the two domains.

4.1. Analogy for Process Scheduling

A hard disk is like a residential colony in which we have people living in villas or in building flats. Each normal person is like a program. When he/she becomes ill due to some disease he changes to a patient (process). All such patients call the hospital (O.S.) and a manager at front desk (long-term scheduler) schedules these patients and creates a list of patients (job queue). Also, for each patient a file (PCB) is created which contains patient id (process id) patient state (new, suspended, waiting), patient's appointment time with doctor (program counter) his previous illness history (data in registers), his current medical state, and some current medication (data in open files). The doctor corresponds to the CPU, the long-term scheduler corresponds to the front-desk person, and the short-term scheduler corresponds to a nurse.

The front desk manager (long-term scheduler) schedules a few patients (processes) to come and wait in hospital lobby (list of such patients forms the ready queue). Number of patients called in a specific session (morning or evening) who wait in hospital lobby for their turn depends upon how many patients will the attending doctors examine in a session (degree of multiprogramming). These patients, who are waiting to be scheduled to enter the doctor’s chamber, are scheduled by a nurse (short-term scheduler) who has her own criteria (FIFO, priority basis, shortest job first, round robin) for letting the patient enter the doctor’s (CPU) chamber. A patient enters the doctor’s chamber and is being questioned and examined (CPU executing the process, process in running state). Doctor (CPU) with the help of previous medical history in patient’s file (PCB) examines the patient (process in running state). While doctor is examining, some emergency patient (high priority process) or a senior doctor calls this doctor (O.S. interrupts), then the current patient is asked to go back and wait in lobby (process interrupted) and he/she will be called back once the doctor has attended the emergency patient (high priority process) or attended the call of senior doctor’s call (O.S. generated system call).

The doctor after examining the patient, asks him to gets certain tests done, e.g., blood test, sugar test, etc., and come back with the test results. These tests are done in various rooms of the hospital (I/O taking place). Other patients are also waiting in these I/O queues, their files (PCB’s) arranged in a sequential manner (linked lists of PCB’s); patient waits for his/her test to be over and waits for its report (I/O gets completed or I/O event takes place). Patient goes back to the ready queue and waits for his/her turn to meet doctor (process which was interrupted will get executed for the remaining leftover part of process). Short-term scheduler keeps track of the processes, whether they are new or they are ones referred for tests, and schedules them accordingly. Those who went for tests are given priority over new patients.

For certain patients, the doctor prescribes some tests for which the reports may be available after some days (an I/O event which takes a long time to execute) in such a case the patient (process) is asked to get his test done, go back home (process gets swapped back to the hard disk, freeing the RAM for other processes to come in) and come back once his reports are available with him. This patient is scheduled by a junior doctor or a manager at hospital to come back on the day when his reports are available (I/O process is over or a called subroutine is done), this junior doctor (medium term scheduler that performs a task called swapping back) keeps track of such patients (processes incomplete and swapped back to hard disk) as to when they are scheduled to come back to hospital and get examined

Whenever the doctor is being interrupted by an emergency call from the emergency department, the patient who was being attended by the doctor is temporarily suspended (context switching takes place) and asked to once again wait in the ready queue. Such a patient's file is updated (contents of Process control Block are saved) and removed from the doctors desk for the time being by a nurse or a junior doctor. A new file of the emergency patient is opened up for the doctor (the new processes PCB is opened up) and the doctor starts attending this new patient (this new process is now being executed). Temporarily closing the old file (saving old PCB) and opening up of new file (opening new PCB) in front of the doctor are the activities that are performed by the junior doctor or nurse (kernel of OS) whenever an emergency patient comes to hospital (an interrupt or system call occurs; these are the activities performed by OS during context switching of processes).

As one can notice, the analogy between OS process scheduling and patient scheduling in hospital is extremely close. It greatly helps in understanding the topic, as students are much more familiar with the latter. Unfortunately, due to lack of space, we are not able to include figures that visually show the close correspondence between the two concepts.

4.2. Evaluation

The first author has been teaching CS concepts using analogy-based methods for the last 10+ years. In fact, the various analogies mentioned earlier have been developed, through considerable investment of effort, once the benefit of teaching CS students via analogies was realized. We invested effort in developing analogies that were realistic and structurally strongly aligned. These analogies have been refined over a number of years. Our observation and evaluation confirm that analogy based instruction produces excellent learning outcomes. Students prefer to register for classes of instructors who use analogies.

We have evaluated our analogy-based teaching methods through feedback from students. We sent a questionnaire to all 60 students who were taught operating systems course in autumn of 2020 at our institution. These students are now seniors (final year students), and will graduate in spring of 2021, and thus are free to speak their minds. Two major concepts in this course were taught using analogies: (i) process scheduling (described earlier) and (ii) virtual memory & memory hierarchy. The questions asked were as follows:

1. Q1: The doctor's office analogy helped you understand the topic of "Processes" better.
2. Q2: The library analogy helped you to understand "Virtual Memory" topic better.
3. Q3: You found topics for which an analogy was given easier to understand.
4. Q4: You found topics for which an analogy was NOT used, harder to understand.
5. Q5: You would like all topics in CS to be taught by giving concrete analogies.

Table 1: Summary of Student Responses.

	Q1	**Q2**	**Q3**	**Q4**	**Q5**
Mean	4.9	4.5	4.4	3.6	4.5
Median	5	5	4	3	5
Mode	5	5	4	3	5

Response were designed in the Likert scale (Strongly Disagree = 1, Disagree = 2, Neutral = 3, Agree = 4, Strongly Agree = 5). 32 responses were received. Table 1 shows the mean, median and mode for the five questions. Clearly, students found analogies from everyday life for teaching advanced CS concepts (processes & virtual memory) to be quite useful for understanding (Q1 and Q2). Students showed clear preference for analogy-based instruction (Q3, Q5). These results are consistent with student feedback, who expressed considerable appreciation for being able to understand a complex topic taught via an analogy.

5. Conclusions and Future Work

We reported on our experience teaching advanced CS concepts using analogies. We listed the analogies that we have developed and used over the last 10+ years for teaching core CS concepts. Our qualitative and quantitative evaluations show excellent results with respect to learning outcomes achieved. We plan to continue developing analogies for more core topics, as well as refine the ones that we have already developed. We plan to publish a compendium of these analogies. We would also like to quantitatively establish that analogy-based instruction leads to improved learning outcomes. Thus, in the future we plan to perform more extensive evaluation based on giving exams and having a control group where analogy-based instruction will not be used (though we will re-teach those CS topics to the control group again using analogies, to ensure that they do not miss out on the benefits of analogies).

References

Bartha, Paul (2019). *Analogy and Analogical Reasoning*. The Stanford Encyclopedia of Philosophy (Spring 2019 Edition), Edward N. Zalta (ed.).

Gadgil, S. and Nokes, T. (2009). Analogical scaffolding in collaborative learning. *In Proc. annual meeting of the Cognitive Science Society*, Amsterdam, The Netherlands

Gentner, D. (1983). Structure-Mapping: A Theoretical Framework for Analogy. *Cognitive Science*. 7 (2): 155–170.

Gentner, D. and Colhoun, J. (2010). A*nalogical processes in human thinking and learning. In Towards a theory of thinking*, pages 35–48. Springer, Berlin, Heidelberg.

Glynn, S. (1994). *Teaching Science with Analogies: A Strategy for Teachers and Textbook* Authors. Nat. Reading Res. Ctr. Research Rep.15. https://eric.ed.gov/?id=ED373306

Hofstadter, D. R. (2001). *Analogy as the core of cognition. The analogical mind: Perspectives from cognitive science*, pages 499–538.

Nokes, T. J. and VanLehn, K. (2008). Bridging principles and examples through analogy and explanation. In Proceedings of the *8th international conference for the learning sciences*, Volume 3, pages 100–102. International Society of the Learning Sciences.

Newby, T.J., Stepich, D.A. (1987). Learning abstract concepts: The use of analogies as a mediational strategy. *Journal of Instructional Development* 10, 20–26.

Ruef, K. (2011). *The Private Eye*. Accessed Jan., 2021. http://www.the-private-eye.com/

Wikipedia contributors. (2021). *Analogy*. In Wikipedia, The Free Encyclopedia. Retrieved Jan., 2021 from https://en.wikipedia.org/w/index.php?title=Analogy.