

Olympiads in Informatics

13

IOI

INTERNATIONAL OLYMPIAD IN INFORMATICS

ISSN 1822-7732

**INTERNATIONAL OLYMPIAD IN INFORMATICS
VILNIUS UNIVERSITY**

OLYMPIADS IN INFORMATICS

Volume 13 2019

Selected papers of
the International Conference joint with
the XXXI International Olympiad in Informatics
Baku, Azerbaijan, 4–11 August, 2019



OLYMPIADS IN INFORMATICS

Editor-in-Chief

Valentina Dagiene
Vilnius University, Lithuania, valentina.dagiene@mif.vu.lt

Executive Editor

Mile Jovanov,
Sts. Cyril and Methodius University, Macedonia, mile.jovanov@finki.ukim.mk

Technical Editor

Tatjana Golubovskaja
Vilnius University, Lithuania, tatjana.golubovskaja@mif.vu.lt

International Editorial Board

Benjamin Burton, University of Queensland, Australia, bab@maths.uq.edu.au

Sébastien Combéfis, Computer Science and IT in Education NPO, Belgium,
sebastien.combefis@csited.be

Michal Forišek, Comenius University, Bratislava, Slovakia, misof@ksp.sk

Gerald Futschek, Vienna University of Technology, Austria, futschek@ifs.tuwien.ac.at

Marcin Kubica, Warsaw University, Poland, kubica@mimuw.edu.pl

Ville Leppänen, University of Turku, Finland, villelep@cs.utu.fi

Krassimir Manev, New Bulgarian University, Bulgaria, kmanev@nbu.bg

Seiichi Tani, Nihon University, Japan, tani.seiichi@nihon-u.ac.jp

Peter Waker, International Qualification Alliance, South Africa,
waker@interware.co.za

Willem van der Vegt, Windesheim University for Applied Sciences, The Netherlands,
w.van.der.vegt@windesheim.nl

The journal Olympiads in Informatics is an international open access journal devoted to publishing original research of the highest quality in all aspects of learning and teaching informatics through olympiads and other competitions.

<https://ioinformatics.org/page/ioi-journal>

ISSN 1822-7732 (Print)
2335-8955 (Online)

© International Olympiad in Informatics, 2019
Vilnius University, 2019
All rights reserved

Foreword

IOI, the International Olympiad in Informatics, is an annual international informatics competition for individual contestants from over 80 invited countries, accompanied by a one-day scientific conference for delegation leaders, organisers and guests. The IOI community has an excellent opportunity to communicate during this international event. Many countries have a variety of things to present and discuss.

The IOI journal is focused on the research and practice of computing professionals who work in the field of teaching informatics to talented secondary and high school students. The journal is closely connected to the scientific conference annually organized during the IOI. The 13th volume has two tracks: the first section of the journal focuses on research, and the second section is devoted to sharing national experiences. In this volume we include the work of some regular contributors to the journal, and also the work of some new authors.

In his paper, D. Ginat argues that algorithmic problem solving involves a collection of implicit notions, which may be considered as tools, since they are repeatedly utilized in various ways, particularly in challenging algorithmics. He says that often employment of these notions is essential, and they pave the way to a desired solution prior to utilization of algorithms and data structures. T. Verhoeff presents his thoughts on programming, software development, and computer science (CS), and their inevitable relationship – “the Golden Triangle”.

Some of the other papers in this volume deal with teaching programming in primary and secondary schools. M. Dolinsky and M. Dolinskaya introduce an approach on how to begin the process of teaching programming to elementary school pupils, by training them to write simple programs – programs that deal with numbers. According to the authors, this learning stage should follow the two previous stages in the process: mental skills development, and learning the keywords of the programming language taught.

M. Weigend, J. Vanicek, Z. Pluhar and I. Pesek explore the potential of using creative unplugged activities in the classroom for Computational Thinking education. They propose a model which consists of 4 types of creative unplugged activities, and they also present the results of an international survey conducted in 2018, regarding the proposed model. In their papers, S. Combefis, G. de Moffarts and M. Jovanov present a new digital library with resources to teach and learn computer science, and M. Lodi, D. Malchiodi, M. Monga, A. Morpurgo and B. Spieler present a survey on the constructionist attempts at supporting the learning of computer programming.

Set of authors present some valuable results from surveys they conducted in Japan. T. Kakeshita, N. Takahashi, K. Sumi and M. Ohtsuki present comprehensive analyses of

three different nationwide surveys on the status of computing education, conducted at the Japanese universities. The first survey (Kakeshita & Ohtsuki) focuses on informatics in general education, and the second one (Kakeshita, Takahash & Ohtsuki) concerns computing education at non-IT departments in Japan. The third one (Sumi, Ohtsuki & Kakeshita) is on computing education at Japanese universities, as subject of “information” for high school teacher’s license.

Finally, W. van der Vegt and E. Schrijvers present a method on how to analyse task difficulty in a Bebras contest using Cuttle, and M. Jancheski and S. Janceska discuss on multidisciplinary, multilingual, multilevel and multipurpose usage of the GeoGebra software in education.

We understand and support the need for continuing to share our national experiences – our problems are common problems. In the second part of the volume, authors from a few countries presented their experience, news and new approaches.

A report on the organization of the Cyprus Olympiad in Informatics done by P. Eraclidean, P. Pavlikas, A. Ttofari, and A. Charalampous review the contest format used for each of the different age groups, as well as the tools and methods utilized in the process of preparation and selection of the national teams of Cyprus for the international competitions.

M. S. Tsvetkova and V. M. Kiryukhin inform about the need to introduce a new profession in the modern society named Digital Curator – consultant in the field of digital literacy. They explain the role of the digital curator and discuss the competencies in detail.

Representatives from this year’s IOI hosting country, Azerbaijan, Y. Tabesh, S. Zarkesh, A. Zarkesh and I. Fazilova, present their experience on computational thinking in K-12. M. Taki and A. Alnahhas present an annual computer science competition in Syria for children aged between 8 and 15 years, which aims at preparing new generations for the future in computer science.

K. Merjalali, A. Keivan Mohtashami, M. Roghani and H. Zarrabi-Zahed present a tool for developing tasks in programming contests called TPS (Task Preparation System), which was successfully used in IOI 2017, and since then – in several other nationwide and international programming contests. Finally, M. Medvediev discusses on the use of the E-Olymp internet portal in programming competitions, a portal that so far supports four languages (Ukrainian, Russian, English and Azerbaijani).

Many thanks to all of those who have assisted with the volume – especially authors and reviewers as well as the Editorial Board of this journal. A lot of work is required by authors, far before starting to write the paper, and there is a lot of work to be done in the process after submitting the first version of the paper until the final version ready for print. In particular, we would like to thank the organisational committee of this year’s IOI in Baku for giving us the opportunity to host the IOI conference.

Editors

TLCS: A Digital Library with Resources to Teach and Learn Computer Science

Sébastien COMBÉFIS¹, Guillaume DE MOFFARTS¹, Mile JOVANOVIĆ²

¹*Computer Science and IT in Education ASBL, Louvain-la-Neuve, Belgium*

²*Faculty of Computer Science and Engineering, Ss. Cyril and Methodius University
st. Rugjer Boshkovikj 16 Skopje, North Macedonia*

e-mail: sebastien@combefis.be, guillaume.demoffarts@csited.be, mile.jovanovic@gmail.com

Abstract. Nowadays, teaching and learning computer science is done at various ages, for several topics and for different reasons. Depending on the country, it can start from the primary school and it finishes at the higher education level, or even later if we take continuing education into consideration. Topics to be learned can be as simple as binary representation or basic programming concepts that can be taught to children to introduce them to computer science. It is also possible to teach and learn advanced data structures or algorithms optimisation, which are interesting skills for Olympiad in Informatics contestants, for example. Recently, there is a prominent number of websites and applications that have been created to help the teaching and learning of many informatics concepts. This paper presents a platform that has been designed to browse a database of resources that can be used to teach or to learn computer science. This digital library contains freely accessible resources and can be searched efficiently thanks to the proposed structure for its content. It has been designed to maximise the user's experience and to fit modern models of digital libraries. For each resource, a detailed information sheet has been produced, containing among other things pedagogical information to help teachers and learners use the resources as best as possible. This platform can also be used to train candidates to Olympiad in Informatics and other related and similar competitions.

Keywords: computer science education, digital library, pedagogical resources database, teaching and learning.

1. Introduction

Computer science is everywhere today and a lot of people are either teaching or learning some of its concepts. In some countries, computer science education starts with very young pupils still in primary school (6–12 years old), and continues with secondary school pupils (12–18 years old). Unfortunately, computer science, or just computational thinking, as a separate subject in curricula is still not widespread in the world. Hopefully, some studies on how computer science could be introduced in curricula are being

conducted (Angeli *et al.*, 2016; Barr *et al.*, 2011; Webb *et al.*, 2017). For contestants to the International Olympiad in Informatics, or other related contests, some skills of computer science are also very important, such as programming and algorithm design, for example. It is not always easy to learn computer science concepts since it often requires high level skills such as abstraction, algorithmic thinking capabilities, creative thinking, etc. Hopefully, and interestingly thanks to informatics, a lot of tools have been designed to help learning computer science concepts, and related skills.

There are a lot of tools that can be used to teach and learn computer science concepts, which are often developed as websites or applications. One main issue is that they are not always well advertised or are not so easy to find. Also, some of them being research prototypes, they often lack documentation or advices on how they can be used to support learning. The work presented in this paper tries to tackle this issue by proposing a database gathering websites and applications to teach and learn computer science.

The proposed database has been developed as a digital library, a tool that can be defined as “*a set of electronic resources and associated technical capabilities for creating, searching and using information*” (Borgman, 1999). The paper presents an online platform that has been designed as a frontend for this database, to allow teachers and learners to quickly find resources relevant to them, and to get information about how to use these resources. Since digital libraries are typically “*constructed, collected and organised by (and for) a community of users*” (Borgman, 1999), this work also proposes future extensions of the online platform to make it easier to grow and involve the community of computer science educators.

The next section presents some related work on digital libraries, their design and how to make them efficient. Section 3 then presents how the database has been structured and how the classification of the resources of the proposed digital library has been done in the frame of this work. Section 4 describes the online platform that has been developed and shows its main features. Finally, the conclusion discusses on the advantages of the proposed platform and presents the future directions that are envisioned for this digital library.

2. Related Work

Digital libraries research emerged in the early 1990s, mainly to identify how they can help and contribute to education. A digital library can be seen as a set of resources that are organised in some way to offer services to its users. Digital libraries definitely play multiple roles in teaching and learning. In particular, Marchionini *et al.* (1995) highlights three main roles: sharing resources, preserving and organising ideas and bringing together people and ideas. Moreover, digital libraries target users with different needs: formal, informal and professional learning missions. Borgman (1999) adds another dimension to the definition of digital libraries, pointing out that they can be seen as content collected on behalf of user communities for researchers and as institutions or services for librarians. More recently, Blandford (2006) focused the interaction role between users

and information that digital libraries convey, allowing for their users to find and to work with the content of the digital library.

Several digital libraries have been developed in the particular case of computer science education. Fox (1996) developed a digital library to increase the quality of learning about computer science. In the frame of his project, several changes have been made at Virginia Tech, mainly concerning the infrastructure, the pedagogy, the evaluation and the tools. The conclusion of his experiment shows that students are learning new topics in a new way, making them happy with the digital library whose number of accesses got a growth for both local and remote access. More recently, Tungare *et al.* (2007) created a syllabus repository of computer science courses across universities in the USA, with as goal to provide added value to the computer science education community. The features provided by this digital library include classifying syllabi, assisting instructors when they are creating new syllabi, and allowing the community to share their syllabi automatically and to compare syllabi for similar courses. Both these works are focused on computer science courses related content.

Today, there are a lot of online resources that can be used to learn programming and other computer science topics. At first, we may think about Open Educational Resources (OER) or Massive Open Online Courses (MOOC) that were made possible thanks to the tremendous growth of ICT in recent years, opening up new opportunities for education, and accessible ways to enjoy quality teaching and learning at all levels (Jemni *et al.*, 2017). In addition to these resources, most of the time associated to courses, people can also learn a lot through programming contests, such as Olympiads in Informatics and other related programming competitions (Combéfis *et al.*, 2014), or with games (Combéfis *et al.*, 2016). The approaches presented in the two latter papers follow the current trend of new models of open and distributed learning (Downes, 2017). As summarised by the author, the important changes are the fact that the learner must go from passive to active and from formal to informal, which is possible thanks to open and distributed learning.

Distance-based education fostered the development of educational tools that can be used online to support teaching and learning. Some of these tools have been developed for MOOCs, such as code executors and graders (Combéfis *et al.*, 2015; Bey *et al.*, 2018) and graph sketchers (French *et al.*, 2017), for example. Other tools are stand-alone applications that can be used independently, online or just locally after installation. All these tools have been developed thanks to computer science, and improve the learners' experience.

For the particular case of computer science education, there are also a lot of tools and prototypes that have been thought about and developed by researchers. For example, Combéfis *et al.* (2013) presented a tool with interactive problems that can guide learners from the understanding of the problem to the coding of a solution for it. Another example comes from Guo (2013) who developed a tool to visualise the execution of any program for learners to map static textual representation (source code) to what is dynamically happening in the computer (execution). A last example, designed by Folland (2016), is a tool to visualise the execution of SQL INSERT statements to highlight how their results are built from source tables.

As mentioned above, another useful resource to learn programming, and other computer science topics, is games (Combéfis *et al.*, 2016). They are playing a large role in teaching computing in higher education, as testified by some reviews. For example, Batistella *et al.* (2016) pointed out that several computing knowledge areas are covered by games, software engineering and programming fundamentals being the most common covered fields. Nevertheless, games are not a panacea as highlighted by Rondon *et al.* (2013), whose study showed that compared to traditional learning, games are only interesting for short-term knowledge retention, at least for medical education. At least, games help to get learners involved with the learning activities, as reported by Schmitz *et al.* (2011), following their experiment.

The examples of tools just presented show that more and more resources are being designed to help the learning of several topics in computer science, namely programming, database, algorithm thinking, etc. Nevertheless, it is not always easy to find such resources. Grissom *et al.* (1998) highlighted the need for a digital library of computer science teaching resources, years ago. More recently, Dichev *et al.* (2012) explained that looking for an appropriate resource is a frequent activity in the job of teaching. Whereas digital libraries of OERs do exist, in particular in the context of courses, no such digital library seems to exist for more general tool resources that cover various computer science topics.

To be efficient, usable and useful, a digital library must be well designed, in particular in the frame of education. Sumner *et al.* (2003) highlighted several key factors that influence the perception of educators about the quality of digital libraries, when used for education. The main results show that a good digital library should favour resources (1) that encourage active learning, (2) that do not result in any bias regarding political or commercial orientations, (3) that limit the access to resources with advertising, (4) that are usable and well-designed to ease the navigation and usability, (5) and that avoid any distractions affecting the attention of learners. These observations have been pointed out by learners as well as by teachers.

Other studies have been made about the interface of digital libraries. In particular, Druin *et al.* (2001) put a focus on this need, especially for children that do not want to just search for information, but also need to use it and need a reason to browse for an item. Compared to a traditional library, a digital library may lack social interaction. Ackerman (1994) insists that social exchanges and interaction are important. The design and use of a digital library should not be limited to the technical mechanisms and the access of information. Mechanisms to make these social interactions possible and to foster them should therefore be thought about. Gazan (2018) goes one step further to include content creators, in addition to content consumers, as an important set of users of digital libraries. The author highlighted the possibility to include user-generated content into digital collection items, therefore increasing the social interactions. Finally, Sumner *et al.* (2004) analysed three models that can be used as approaches to educational digital library design. Their conclusion is that digital libraries can be used (1) as cognitive tools to support learning and help users to catch the sense of the activities, (2) as component repositories to focus on how resources from the digital library

are produced and distributed, (3) and as knowledge networks to foster social interactions and knowledge building and sharing.

The different elements highlighted by these related works have all been somewhat taken into account for the design of the platform presented in this paper.

3. Classification of Websites and Applications

The websites and applications that can be used to teach or to learn computer science are classified according to several criteria. The proposed characterisation is meant to help teachers and learners to choose the most suited and relevant website or application that fits their needs. It should also help teachers to use the resources in the most effective way. For the platform to be powerful, yet flexible, and to ease the development of a community of users around the platform resources, a detailed characterisation of the resources is proposed in this paper and explained in this section.

3.1. Category


The first classification criterion is related to the kind of service that the website or the application is providing. According to the resources that have been considered and analysed in the frame of this work, six main categories have been identified:

- Directory.
- Visualiser.
- Animated tutorial.
- Playground.
- Interactive tutorial.
- Game.

3.1.1. Directory

The *directory* category gathers resources that allow their users to navigate through a collection of resources, technologies, tools, etc. Websites or applications from this category help learners to discover resources related to the same topic or field of study. For example, the “*NoSQL Databases*” website, shown on Fig. 1, maintains a large list of NoSQL databases engines organised according to their main paradigm. It is an interesting resource for anyone who discovered the NoSQL world and wants to explore the existing engines or to choose one for a project.

Resources from this category are similar to the “awesome list” movement whose main goal is to gather a curation of awesome stuff in lists (Sorhus, 2019). The main difference between a simple collection or aggregation and a curation is that the latter involves a selection of content based on quality (Dale, 2014). For example, Caiero-Rodríguez *et al.* (2013) proposed a social curation platform for OERs.



Your Ultimate Guide to the
Non-Relational Universe!

[including a historic [Archive](#) 2009-2011]
News Feed covering some changes [here](#) !

NOSQL DEFINITION:Next Generation Databases mostly addressing some of the points: being non-relational, distributed, open-source and horizontally scalable.

The original intention has been modern web-scale databases. The movement began early 2009 and is growing rapidly. Often more characteristics apply such as: schema-free, easy replication support, simple API, eventually consistent / BASE (not ACID), a huge amount of data and more. So the misleading term "nosql" (the community now translates it mostly with "not only sql") should be seen as an alias to something like the definition above. (based on 7 sources, 15 constructive feedback emails (thanks!) and 1 disliking comment. Agree / Disagree? [Tell me so!](#) By the way: this is a strong definition and it is out there here since 2009!)

LIST OF NOSQL DATABASES [currently >225]

Core NOSQL Systems: [Mostly originated out of a Web 2.0 need]

Wide Column Store / Column Families

[Hadoop / HBase](#) API: Java / any writer, Protocol: any write call, Query Method: MapReduce Java / any exec, Replication: HDFS Replication, Written in: Java, Concurrency: ?, Misc: Links: 3 Books [[1](#), [2](#), [3](#)], [Guru99 Article](#) >>

[MapR](#), [Hortonworks](#), [Cloudera](#) Hadoop Distributions and professional services .

[Cassandra](#) massively scalable, partitioned row store, masterless architecture, linear scale performance, no single points of failure, read/write support across multiple data centers & cloud availability zones. API / Query Method: CQL and Thrift, replication: peer-to-peer, written in: Java, Concurrency: tunable consistency, Misc: built-in data compression, MapReduce support, primary/secondary indexes, security features. Links: [Documentation](#), [PlanetC*](#), [Company](#).



[Scylla](#) Cassandra-compatible column store, with consistent low latency and more transactions per second. Designed with a thread-per-core model to maximize performance on modern multicore

NoSQL RELATED EVENTS:

- June 26-27 2018 MongoDB World [»](#)

Register your event 4free: [»](#)

NoSQL ARCHIVE

the multi-model NoSQL DB

NoSQL FORUMS

- Global NoSQL Forum [»](#)
- Forum Berlin [»](#)
- Forum France [»](#)
- Forum Japan [»](#)

NoSQL NEWS FEEDS

- MyNoSQL by Alex P [»](#)
- On Twitter: nosqlupdate [»](#)
- NoSQL Weekly [»](#) * new *
- HighScalability Blog [»](#)

Fig. 1. The *NoSQL Databases* website maintains a collection of NoSQL database engines that are organised according to their main paradigm.

3.1.2. Visualisation

To teach and to learn new concepts, it can help a lot to be able to visualise, in some way, the new concepts. In particular, people who are more sensitive to visual modalities will benefit from such visualisations. The second and third categories contain resources that propose tools to visualise concepts.

The *visualiser* category contains tools that can produce visualisations, either static or dynamic ones. The goal of these visual elements is to help you to represent yourself the concepts you are supposed to learn. Such tools can also be used for teaching purpose, to provide visual examples to your students (Fouh *et al.*, 2012). As highlighted by Naps *et al.* (2002), visualisation is only effective if it engages learners in an active learning activity. It is therefore important to provide explanations on how the use the visualisation tool to support learning.

For example, the “*viSQLizer*” platform (Folland, 2016), shown on Fig. 2, is a prototype visual learning tool for SQL. The tool builds and shows animations to illustrate how the result of a SELECT query is built by extracting rows from the involved tables. This can be used in an introductory course on databases and queries, to illustrate how data

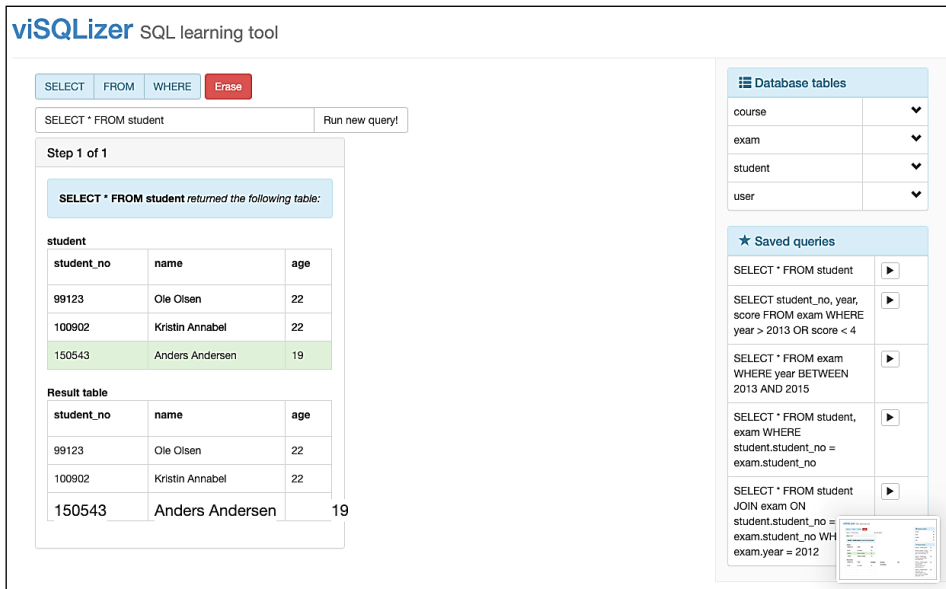


Fig. 2. The *viSQLizer* prototype visual learning tool helps learners to understand how the result of a SELECT query written in the SQL language builds its result from tables.

organised in tables can be scanned through to get the result of a given request. It is also interesting to see the different steps behind a SELECT query, starting with row filtering followed by column projection.

Resources from this category, if correctly used, will engage their users in their own learning. This exactly matches good visualisation tools as defined by Naps *et al.* (2002). Unfortunately, most of them being the result of PhD or master thesis, they lack pedagogical documentation on how to use them effectively. Also, they are not accompanying the learner within a learning path.

The *animated tutorials* category is dedicated to websites and applications that provide a tutorial meant to teach new concepts, by presenting you direct examples with the produced results (Rodger, 2002). It goes one step further compared to the visualiser category, in the sense that the visualisation are embedded within a tutorial that guides you for your learning. For example, the “*Unfolding the Box Model*” website, shown on Fig. 3, shows you how do CSS 3D transforms work. Each page of the tutorial just shows you directly the result of the transforms that are presented.

Visualisation tools allow the user to ask for a visual representation of a given input, such as an SQL query, an operation on a given data structure, an execution of an algorithm for a problem instance, etc. Once the input has been provided, the tool shows a visualisation that the user is just watching. In the case of animated tutorials, the user is presented a sequence of visual animations to gradually explain the user concepts, like a tutorial would have done.

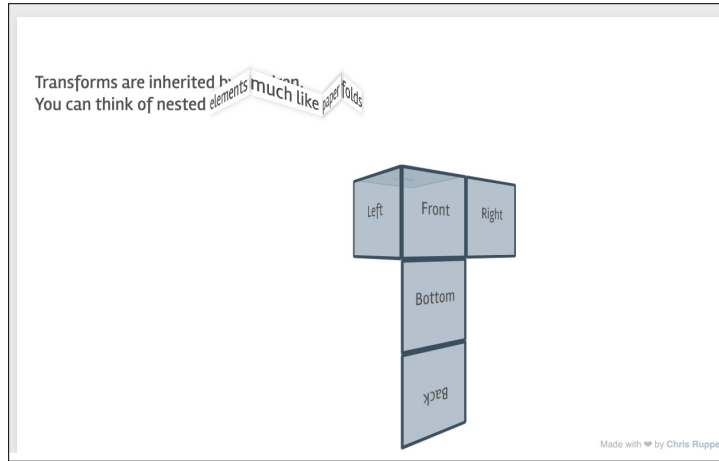


Fig. 3. The *Unfolding the Box Model* website shows you how CSS 3D transforms work with concrete examples that have been put as a single animated tutorial.

3.1.3. Interaction

The next two categories add the ability for the user to interact actively with the website or the application. The user is offered the possibility to play with his/her own examples and gets a direct feedback. In this way, the user can be challenged and put in the centre of his/her own learning, which will contribute to improve what he/she will learn and assimilate (Bork, 2001).

The *playground* category is for websites and applications where the user can enter codes, problem instances, situation descriptions, etc. and execute them to visualise and get the result directly. Such resources are useful for the user to be able to play without the need to install anything on his/her computer. For example, the “*RxViz*” website, shown on Fig. 4, allows you to play with RxJs observables in an animated playground. It makes it possible for you to write your own code, or even to take one of the proposed examples, to execute it and to get the result in a visual way.

Playgrounds are very similar to simple visualisation tool, except that they provide more freedom and can visualise much more complex objects, especially code. These tools are showing a visual execution of the code along with the execution of the latter. Changing the code and executing it again will directly update the visualisation. Sometimes, it is also possible to directly interact with the visualisation, and the code could be updated accordingly.

The *interactive tutorial* category is one step further the animated tutorial, in the sense that the user will be challenged and asked to interact with the animations. As a tutorial, it is accompanying the learner during the learning process. And as an interactive tutorial, it asks the learner to take part to the learning process through different kinds of interactions. For example, the “*Computer Science Field Guide*” is an online interactive book that can be used to teach various computer science concepts to high school students. It provides interactive exercises throughout the book that allow the learners to experiment what they learned. Fig. 5 shows one of the interactive exercises that are proposed on the website.

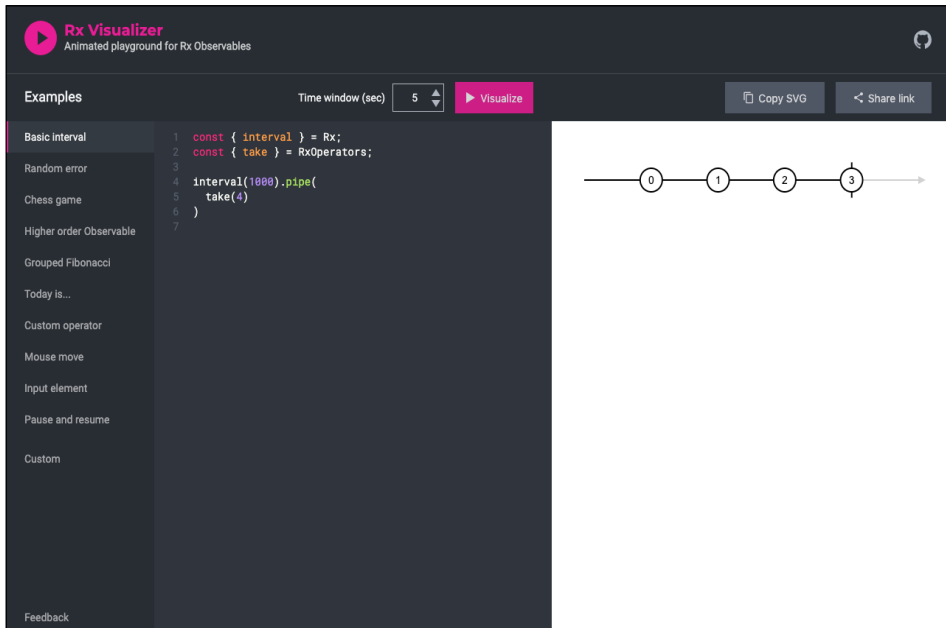


Fig. 4. The *RxViz* website is a playground where you can write and execute programs using RxJS observables and get a visual interpretation of the result.

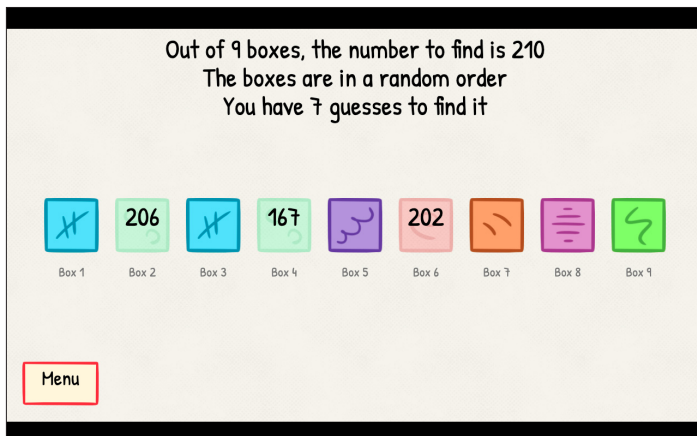


Fig. 5. The *Computer Science Field Guide* is a website that proposes interactive exercises, as part of an online book, to help its learners to understand the new concepts.

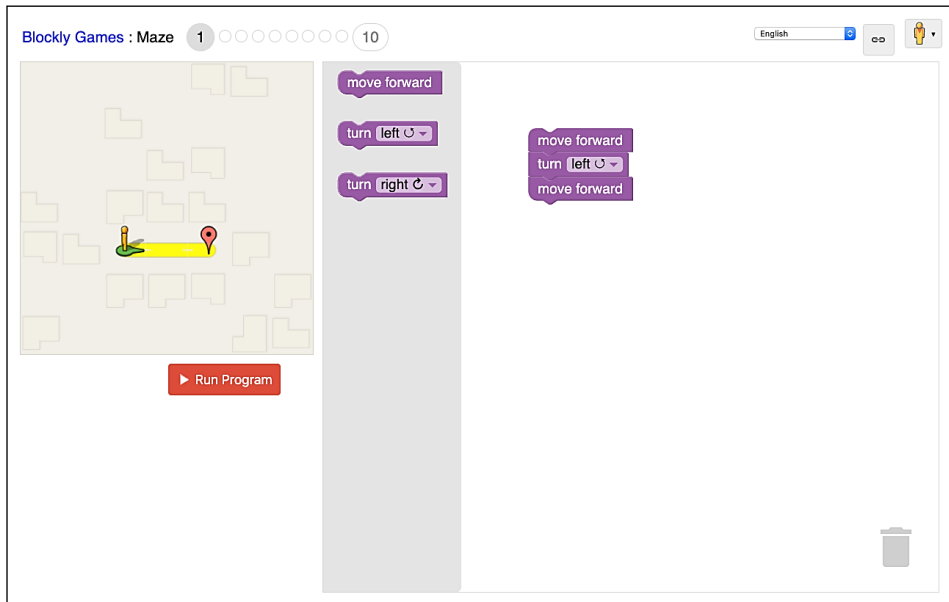


Fig. 6. The *Blockly Games* website proposes a game with several tasks that have to be solved using a blocs-based visual programming language in order to win the game.

3.1.4. Game

Finally, websites and applications from the last category, namely *game*, provide the most interactive experience to the learners and require them the largest involvement. It also tries to motivate them with the addition of goals, scoreboards, competition, etc., compared to the interactive tutorials (Combéfis *et al.*, 2016).

For example, the “*Blockly Games*” website challenges its users by asking them to solve several tasks whose solutions are programs written with a block-based visual programming language, similar to Scratch. Each task can be seen as a small game, each of these being one level in a bigger game. One of the tasks being solved is shown on Fig. 6.

3.1.5. Overlapping Categories

The six categories presented in this section are overlapping, meaning that some websites or applications can belong to more than one category. For example, the “*RxViz*” website is at the same time a playground and a visualisation tool since it allows you to write and execute any code but presents visually the result of the execution. Another example is the “*SQL Island*” website, presented in section 4, which is at the same time a game and an interactive tutorial. The platform described in this paper allows you to easily navigate the resources database according to the categories.

3.2. Language and Field

The two others classification criteria are the programming *languages* and the computer science *fields* that are covered by the website or application. The first criterion is optional and the second one is mandatory. A computer science resource is indeed always related to at least one field but does not always concern a programming language. The presented platform makes it possible to search for teaching and learning resources based on these two criteria.

The possible values for the programming language criterion are quite clear but it is less obvious for the computer science field. In this work, general fields such as database, programming, data structure, artificial intelligence, etc. have been used. Another possibility would be to use the *ACM Computing Classification System (CCS)*, but it may be too complex for the targeted users for the proposed platform.

Classification by categories, programming language and computer science fields can help to better identify and attract people from existing communities of interest. For example, people interested in resources related to the Python programming language, for machine learning, could easily find them with the proposed platform.

3.3. Level

Teaching and learning computer science is done at various ages and level of education. For each website and application available in the platform described in this paper, the most suited age *levels* are indicated. Five levels have been identified:

- Children goes until 12 years old, that is the end of primary school.
- Junior goes from 12 to 15, that is, lower secondary school.
- Senior goes from 15 to 18, that is, higher secondary school.
- BSc is for bachelor students.
- MSc is for master students.

The identified levels are indicative and correspond to the most suited age groups with which the resource can be used to teach or to learn the concepts conveyed by the resource. This way to organise the resource is directly related with the community of teachers. For example, a primary school teacher will indeed first search for resources relevant for the age of his/her pupils.

3.4. Pedagogical Information

Finally, to help teachers and learners to use the resources available on the proposed platform, various *pedagogical information* can be added to each website or application. Their purpose is to propose a guide to use the resource as best as possible. Three kinds of information can be provided: prerequisite, learning outcomes and methodology.

The *prerequisites* summarise what knowledge should be mastered to be able to use the resource to learn the conveyed concepts. The content of this section should be written according to the proposed age levels. The *learning outcomes* list what the student will be able to do after he/she used the resource in the frame of a learning activity. This section can also contain information about the content proposed by the resource. Finally, the *methodology* explains how the resource can be used or how it is supposed to be used by its original creators and designers.

The three kinds of information are of course not relevant for all the resource categories. For example, pure playgrounds will typically lack learning outcomes and methodologies. Also, learning outcomes could depend on how the resource is used by a teacher in an activity. On the platform presented in this paper, the proposed pedagogical information is written to be consistent, meaning that the learning outcomes are to be read with the proposed methodology on how to use the resource.

4. Interactive Platform

This paper proposes an online platform, called TLCS for “*Teaching and Learning Computer Science*”, available online at the following address: <https://tlcs.csited.be>. It is only available in English for now but is ready for internationalisation. Fig. 7 shows the page describing the “*SQL Island*” website, a game to learn the fundamentals of the SQL database querying language (Schildgen, 2014).

4.1. Structure of the Platform

The layout of the page is structured in three columns. A navigation tool is available on the left part to allow the user to browse the resources by categories, programming languages, computer science fields or levels of education. It is also possible to make some cross-searches by clicking on the magnifying glass and selecting the tags you are interested in. For example, you could search for resources that are interactive tutorials in the form of games, such as illustrated on Fig. 8.

An information panel is visible on the right part to show all the categories, programming languages and computer science fields of the resource. You can also directly see the levels of education and access the website of the resource through this information panel. Depending on the resource, some of the information may or may not be available. Fig. 9 shows the information available for “*SQL Island*”.

Finally, the central column shows a short description with screenshots directly followed by the pedagogical information. Two last optional sections can be available, depending on the resource. The *service* section describes the kinds of service provided by the resource, such as cooperative game, API, possibility to save or share, etc. The *references* section provides scientific references to papers presenting the website or application, when available.

TLCS v1.0
en ↕ About

Category Language Field Q

- ▶ Interactive tutorial
- ▶ CSS Diner
- ▶ SQL Island
- ▶ Grid Garden
- ▶ Game
- ▶ Playground
- ▶ Animated tutorial

SQL Island

SQL Island is an **adventure game** where the hero is stuck on an island and tries to escape from it. In order to communicate with the inhabitants of the island, you have to speak the **SQL database querying language**, the only language they can understand. Playing the game will teach you the fundamentals of SQL in an entertaining and funny way.



It is possible to switch the language to English through the third menu item of the options menu.

Information

- ▶ **Categories**
Game, Interactive tutorial
- ▶ **Fields**
Database, Programming
- ▶ **Levels**
Senior, BSC
- ▶ **Languages**
SQL
- ▶ **Website**
SQL Island

Prerequisites

To be able to play this game, you just need to understand how data can be organised in a tabular way, such as in a spreadsheet, for example.

Learning outcomes

After you escaped from the island, that is, you successfully finished the game, you will be able to understand the **fundamentals of the SQL** database querying language. More precisely, you will discover:

- *SELECT statements* to extract rows from a table;
- *WHERE clauses* to restrict the rows affected by queries on a table and *AND, OR and LIKE operators* to build complex conditions;
- *INSERT INTO statements* to insert rows in a table;
- *NULL values* to indicate missing values for some columns in a row;
- *UPDATE statements* to update existing rows from a table;
- *ORDER BY clauses* to order the extracted rows in ascending or descending order;
- *Natural join operations* to extract rows by combining rows extracted from several tables;
- *COUNT function* to count the number of rows, *SUM function* to sum the values of a column and *AVG function* to compute the average value of a column, for all the rows satisfying the criteria of the WHERE clause;
- *GROUP BY clause* to group rows having the same value for some columns together, optionally used with aggregate functions such as COUNT, SUM, etc.

Methodology

During the game, you will first have to **understand queries** that you want to perform on the villages of the island, their inhabitants and the items they have. After that, you will have to **write them using SQL**. To help you, the system will show you some queries with the results they produced, from time to time.

The game is **incremental** and each step teaches you a new construct. Moreover, **some hints** about which construct you should use for the query you have to perform are sometimes provided. Finally, you can **try any SQL query** at any time and see the produced result, with a comment to help you if you have the wrong answer.

Services

The website provides you a game that you have to **play in one go**. You can also restart the game at any time, if you deleted some rows by mistake, for example.

References

- Schildgen J., & DeBloch S. (2015). SQL-Grundlagen spielend lernen mit dem Text-Adventure SQL Island. In Seidl, T., Ritter, N., Schöning, H., Sattler, K.-U., Härdter, T., Friedrich, S. & Wingerath, W. (Eds.) *Datenbanksysteme für Business, Technologie und Web (BTW 2015)* (pp. 687-690). Bonn: Gesellschaft für Informatik e.V.
- Schildgen J. (2014). SQL Island: An Adventure Game to Learn the Database Language SQL. In *Proceedings of the 8th European Conference on Games Based Learning (ECGBL 2014)* (pp. 137-138).
- Schildgen J., & DeBloch S. (2013). „Gib mir so viel Gold, wie die Metzger im Nachbardorf zusammen besitzen und ich lasse den Piloten frei!“ – Spielbasiertes Lernen von SQL-Grundlagen. *Datenbank Spektrum*, 13(3), 243-249.

Fig. 7. Each website or application is described with a complete information sheet that contains categorisation and pedagogical information.

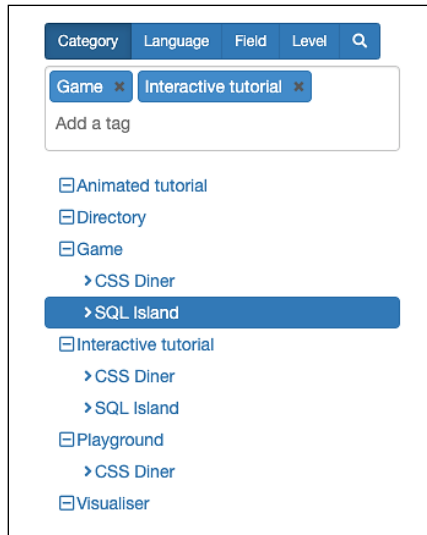


Fig. 8. The left part of the TLCS platform allows you to browse all the resources by categories, programming languages, computer science fields or levels of education. You can also search for resources by tags.

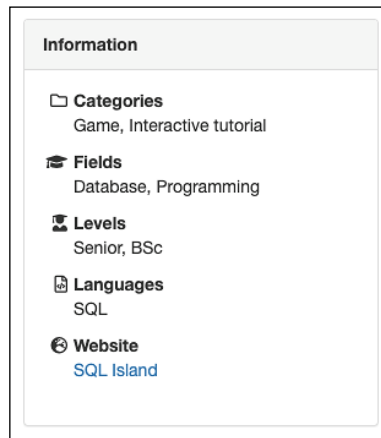


Fig. 9. The right part of the TLCS platform shows you all the available information on the resource you are looking at, allowing you to quickly characterise the resource.

The online platform has been designed to be simple to use and so that the information is clearly presented. It is also very light and runs in any modern browser since it relies on the recent versions of JavaScript. It has been developed with Angular.js for the frontend part and with Bootstrap 3 for the style. The database is just a simple JSON file, in fact one for each language (only English and French being available for now).

4.2. Database Population

The proposed digital library is meant to be populated by the community. It is indeed a way to ensure coherence between the needs of the community and the content offered on the platform. For a digital library to be good, and to ensure that its content is of high quality, some review and/or control mechanism must be put in place. Entries of the database for each resource must be correct, with exact, complete, relevant and up-to-date information.

To achieve the latter requirements, the proposed platform has a public page that anyone can use to propose a new resource for addition. Each proposition has to be reviewed, and is possibly corrected, before being accepted. The *Computer Science and IT in Education* non-profit organisation is currently in charge of this control and acceptance process. It may be opened to new partners in the future, especially when content in other languages than English will be made available. This way to proceed ensures good quality content while keeping the platform somewhat open to the community and its users.

4.3. Social and Community Aspects

As presented in the related work section, social aspects are very important for a digital library to be useful, used and for it to support learning and also knowledge sharing. Some elements to foster social interactions and to highlight community aspects have been thought about for the proposed platform, even if not implemented in the current version yet.

Users will be able to have an account on the platform and decide of their own tags for the resources. This feature allows them to organise the resources with their own categorisation. The second feature is the ability for the users of the platform to grade each resource with stars, so that the best resources will get more stars than the less good resources.

5. Conclusion

To conclude, this paper presents a digital library with websites and applications that can be used to learn computer science concepts. This database is structured so that to be easily queried to find useful and relevant resources to teach or to learn new concepts. Its design and the way its information is structured has been thought about regarding advices about how to make an efficient digital library.

The paper proposes a multi-criteria categorisation of all the resources contained in the database. To help people to search through the database, an online platform has been developed and made available to the community. It proposes a simple yet powerful and

ergonomic interface to look at the resources from the database. This interface brings several intuitive and useful ways to extract relevant information from the developed database. It has still to be improved, in particular to take into account the content creators and to include user-generated content.

Compared to other kinds of digital library that exists, which are mostly focused on OERs for teachers, and in particular for higher education, this work proposes a database that can also be used by learners, from the youngest ones to adults who already graduated. The proposed digital library, which already contains about twenty resources, supports the creation, the search and the use of resources. The information that is provided with each resources supports learning, in the most efficient way as possible.

Future work includes the translation of the platform in several languages as well as the translation of the database content, to widen the community that could therefore enjoy the available data. New resources will also be added, especially applications that can be used on smartphones. Finally, the platform and its interface will also be improved, with the possibility to add comments and notes for each resource, for example. Last but not least, surveys must be conducted with teachers, to evaluate whether the proposed platform fits their needs.

References

- Ackerman, M.S. (1994). Providing Social Interaction in the Digital Library. In: *Proceedings of the 1st Annual Conference on the Theory and Practice of Digital Libraries*. 198–200.
- Angeli, C., Voogt, J., Fluck, A., Webb, M., Cox, M., Malyn-Smith, J., Zagami, J. (2016). A K-6 Computational thinking curriculum framework: Implications for teacher knowledge. *Educational Technology & Society*, 19(3), 47–57.
- Barr, V., Stephenson, C. (2011). Bringing computational thinking to K-12: What is involved and What is the role of the computer science education community? *ACM Inroads*, 2(1), 48–54.
- Batistella, P., von Wangenheim, C.G. (2016). *Games for Teaching Computing in Higher Education – A Systematic Review*.
- Bey, A., Jermann, P., Dillenbourg, P. (2018). A comparison between two automatic assessment approaches for programming: An empirical study on MOOCs. *Educational Technology & Society*, 21(2), 259–272.
- Blandford, A. (2006). Interacting with information resources: digital libraries for education. *International Journal of Learning Technologies*, 2(2/3), 185–202.
- Borgman, C. (1999). What are digital libraries? Competing visions. *Information Processing and Management*, 35(3), 227–243.
- Bork, A. (2001). Tutorial learning for the new century. *Journal of Science Education and Technology*, 10(1), 55–71.
- Caeiro-Rodríguez, M., Pérez-Rodríguez, R., García-Alonso, J., Manso-Vázquez, M., Llamas-Nistal, M. (2013). AREA: A social curation platform for open educational resources and lesson plans. In: *Proceedings of the 2013 IEEE Frontiers in Education Conference (FIE)*. 795–801.
- Combéfis, S., Van den Schrieck, V., Nootens, A. (2013). Growing algorithmic thinking through interactive problems to encourage learning programming. *Olympiads in Informatics*, 7, 3–13.
- Combéfis, S., Wautelet, J. (2014). Programming trainings and informatics teaching through online contest. *Olympiads in Informatics*, 8, 21–34.
- Combéfis, S., Paques, A. (2015). Pythia reloaded: An intelligent unit testing-based code grader for education. In: *Proceedings of the 1st International Workshop on Code Hunt Workshop on Educational Software Engineering (CHESE 2015)*. 5–8.
- Combéfis, S., Beresnevičius, G., Dagienė, V. (2016). learning programming through games and contests: Overview, characterisation and discussion. *Olympiads in Informatics*, 10, 39–60.
- Dale, S. (2014). Content curation: The future of relevance. *Business Information Review*, 31(4), 199–205.

- Dichev C., Dicheva, D. (2012). Open Educational Resources in Computer Science Teaching. In: *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education (SIGCSE 2012)*. 619–624.
- Downes, S. (2017). New Models of Open and Distributed Learning. In: *Open Education: from OERs to MOOCs*. Berlin/Heidelberg: Springer-Verlag, 1–22.
- Druin, A., Bederson, B.B., Hourcade, J.P., Sherman, L., Revelle, G., Platner, M., Weng, S. (2001). In: *Proceedings of the 1st ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL 2001)*. 398–405.
- Folland, K.A.T. (2016). viSQLizer: Using visualization for learning SQL. In: *Proceedings of the 29th Norsk Informatikkonferanse (NIK 2016)*.
- Fouh, E., Akbar, M., Shaffer, C.A. (2012). The role of visualization in computer science education. *Interdisciplinary Journal of Practice, Theory, and Applied Research*, 29, 95–117.
- Fox, E.A. (1996). Interactive learning with a digital library in computer science. In: *Proceedings of 26th Frontiers in Education Annual Conference (FIE 1996)*. 415–419.
- French, J., Segado, M.A., Ai, P.Z. (2017). Sketching graphs in a calculus MOOC: Preliminary results. In: *Frontiers in Pen and Touch*. Cham: Springer, 93–102.
- Gazan, R. (2008). Social annotations in digital libraries collections. *D-Lib*, 14, 11/12.
- Grissom, S., Knox, D. Copperman, E., Dann, W., Goldweber, M., Hartman, J., Kuittinen, M., Mutchler, D., Parlante, N. (1998). Developing a digital library of computer science teaching resources. In: *Working Group Reports of the 3rd Annual SIGCSE/SIGCUE ITiCSE Conference on Integrating Technology into Computer Science Education (ITiCSE-WGR 1998)*. 1–13.
- Guo, P.J. (2013). Online Python tutor: Embeddable Web-based program visualization for CS education. In: *Proceedings of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE 2013)*. 579–584.
- Jemni, M., Kinshuk, Khribi, M.K. (2017). *Open Education: From OERs to MOOCs*. Berlin/Heidelberg: Springer-Verlag.
- Marchionini, G., Maurer, H. (1995). The roles of digital libraries in teaching and learning. *Communication of the ACM*, 38(4), 67–75.
- Naps, T.L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S., Velázquez-Iturbide, J.A. (2002). Exploring the role of visualization and engagement in computer science education. In: *Proceedings of Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education (ITiCSE-WGR 2002)*. 131–152.
- Rodger, S.H. (2002). Introducing computer science through animation and virtual worlds. In: *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2002)*. 186–190.
- Rondon S., Sassi, F.C., Furquim de Andrade C.R. (2013). Computer game-based and traditional learning method: A comparison regarding students' knowledge retention. *BMC Medical Education*, 13, 30.
- Schildgen J. (2014). SQL island: An adventure game to learn the database language SQL. In: *Proceedings of the 8th European Conference on Games Based Learning (ECGBL 2014)*. 137–138.
- Schmitz B., Czauderna, A., Klemke, R., Specht, M. (2011). Game based learning for computer science education. In: *Proceedings of the Computer Science Education Research Conference (CSERC 2011)*. 81–86.
- Sorhus, S. (2019). *Awesome*. <https://github.com/sindresorhus/awesome>
- Sumner, K., Khoo, M., Recker, M., Marlino, M. (2003). Understanding educator perceptions of “Quality” in digital libraries. In: *Proceedings of the 3rd ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL 2003)*. 269–279.
- Sumner, T., Marlino, M. (2004). Digital libraries and educational practice: A case for new models. In: *Proceedings of the 4th ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL 2004)*. 170–178.
- Tungare, M., Yu, X., Cameron, W., Teng, G., Pérez-Quinones, M.A., Cassel, L., Fan, W., Fox, E. (2007). Towards a syllabus repository for computer science courses. In: *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*. 55–59.
- Webb, M., Davis, N., Bell, T., Katz, Y.J., Reynolds, N., Chambers, D.P., Syslo, M.M. (2017). *Education and Information Technologies*, 22(2), 445–468.



S. Combéfis obtained his PhD in engineering in November 2013 from the Université catholique de Louvain (UCLouvain). He is currently working as a lecturer at the ECAM Brussels Engineering School, where his courses focus on computer science. He also obtained an advanced master in pedagogy in higher education in June 2014. Co-founder of the Belgian Olympiad in Informatics (be-OI) in 2010, he later introduced the Bebras contest in Belgium in 2012 and at the same time founded CSITEd. This non-profit organisation aims at promoting computer science in secondary schools.



G. de Moffarts is a master student in computer science at Université catholique de Louvain (UCLouvain). He is interested in computer science and electronics, and very curious about engineering and new technologies, such as 3D printing, artificial intelligence and the internet of things. He is also involved in the CSITEd non-profit organisation, taking part on several projects it organises. He was also recently the deputy leader of a Belgian delegation to the IBU Olympiad in Informatics 2019 that was held in Skopje, North Macedonia.



M. Jovanov is an associate professor at the Faculty of Computer Science and Engineering, Ss. Cyril and Methodius University, in Skopje. As the President of the Computer Society of Macedonia, he has actively participated in the organization and realization of the Macedonian national competitions and Olympiads in informatics since 2001. He has been a team leader for the Macedonian team at International Olympiads in Informatics since 2006. His research interests include development of new algorithms, future web, and e-education.

Training in Writing the Simplest Programs from Early Ages

Michael DOLINSKY, Mariya DOLINSKAYA

*Faculty of Mathematics and Technologies of Programming, F. Skorina Gomel State University
Sovetskaya str., 104, Gomel. 246019. Republic of Belarus
e-mail: dolinsky@gsu.by, mkugejko@gsu.by*

Abstract. This article describes the author approach for start programming teaching at the primary school, which is sequentially teach to create simplest programs that read some numbers, do necessary calculations and write the answer.

Keywords: primary school, programming teaching, distance learning tools.

Introduction

For many years, the authors have been training schoolchildren of the city of Gomel for olympiads in computer science and programming (Dolinsky, 2013). All work is carried out on the basis of the instrumental distance learning system (<http://dl.gsu.by>, hereinafter referred to as DL), created at the mathematics department of F. Skorina Gomel State University under the leadership of one of the authors (Dolinsky, 2017). At the same time, schoolchildren of increasingly earlier ages are drawn into programming training, and since 2007, learning begins in the first grade. According to the authors (based on the results of the performance of Gomel schoolchildren at republican and international computer science competitions), a rather effective system of education has been created (Dolinsky, 2016).

The two first stages of learning were described in detail earlier: mental skills development (Dolinsky, 2014) and learning the starting keywords for programming in Pascal (Dolinsky and Dolinskaya, 2018). This article describes in detail the authors' approach to the subsequent transition to programming training.

Starting from year 2019 Pascal can't be used at IOI, nevertheless the authors consider the article as very useful for our community for the following reasons. First of all, the article describes approach that can be used for any programming language. In addition, authors think that one can continue to teach programming to 7-9 years old children with existing effective system to provide fast and easy learning for early age children. Finally, our distance learning system provides transfer to C++ teaching for any one (Dolinsky, 2017).

№ 1	№ 2	№ 3
Example of data output -2-0-1-7- <2016>	Example of data entry: 77 70 69 Example of data output t:8(0232)777069	Example of data entry: 12 5 Example of data output 12=5+7 7+10=5+12
	Example of data entry: 71 23 67 Example of data output t:8(0232)712367	Example of data entry: 3 1 Example of data output 3=1+2 2+2=1+3

Fig. 1. The purpose of training.

In Fig. 1, our learning goal is concisely presented – to teach how to write programs, decisive tasks for regional programming competitions for pupils of grades 1–4, besides it presents the tasks of the Olympiad, which took place in April 2017.

In the first task you are required to display two specified sequences of characters. In the second task it is required to enter three numbers and put them in the correct order, adding other symbols to the output. To perform these tasks, it is required to use the read and write operators in the standard way (readln, writeln). Fig. 2 shows the author’s solutions to these problems.

But the third task is already a real Olympiad task. It is not so simple to understand or guess what the program should do. In order to facilitate the understanding of the task and ensure the unambiguity of its formulation, TWO examples of input and output are

№ 1	№ 2
Begin writeln('2-0-1-7-'); writeln('<2016>'); end.	var a,b,c : longint; begin readln(a); readln(b); readln(c); writeln('t:8(0232)',a,b,c); end.

Fig. 2. Author’s solutions to problems 1 and 2.

given, a comparative analysis of which allows you to reliably determine what needs to be done in the task. The conditions of the first 10 tasks of competitions for grades 1–4 historically look like two examples of input-output. Initially, we started from the fact that children could not read confidently and, with this in mind, we formulated tasks. On the other hand, the analysis of the conditions of the tasks presented in such a form, by definition, develops mental skills, brings the task closer to the Olympiad task style, and causes additional interest that increases the motivation to practice.

So, let us reproduce the possible analysis of the condition of the third task given. The analysis is made by a pupil. It is required to type two numbers on the keyboard (12 and 5). Next you need to display the line $12 = 5 + 7$. 12 and 5 are the numbers that were input. The number 7 was not on the input, hence this is either a constant number, or a number that our program needs to calculate. In the second example, the number 5 is not at this position. It means that the program must calculate it. From the example it is clear how to calculate it. This should be such a number, which in total with 5 will give 12. Hence, to calculate it, subtract 5 from 12. Then output, in the correct sequence, the numbers entered, the calculated number, as well as the symbols “+” and “=”.

Now let’s turn to the second line of the sample output. 7 is the number we have just calculated, by the second example we check our guess: there is a number 2 on the corresponding position, also calculated as the difference between the entered numbers 3 and 1. On the right side of the output, after the “equal” sign, there are 5 and 12 that were on input. But in the left part there is also a new number 10. Again, it is clear from the example that this is a number that, in total with the number 7 we calculated earlier, should give the sum of the numbers 5 and 12. Therefore, it should be calculated as the difference $(5 + 12) - 7$. Similarly, for the second example, the right number 2 is calculated as the difference $(1 + 3) - 2$.

The author’s solution of task 3 is shown in Fig. 3.

As for the Olympics, the first such contest took place on October 27, 2008. Training of hundreds of children since that time has led us to a deeper understanding of the problems arising in teaching and to the updating and development of the automatic learning system.

```

var
  s1,s2,s3,s4 : longint;
begin
  readln(s1);
  readln(s2);
  s3:=s1-s2;
  s4:=s1+s2-s3;
  writeln(s1,'=',s2,'+',s3);
  writeln(s3,'+',s4,'=',s2,'+',s1);
end.

```

Fig. 3. Author's solution to problem 3.

Teaching Technology

Thus, the complete list of tasks for learning to write the first program in Pascal (manipulating numbers) seems to us like this:

- Move from words to the text of the first program (input-output numbers, Fig. 4).
- Learn how to work in the Turbo Pascal environment.
- Launch Pascal from the desktop icon.
- Type and edit the program.
- The minimum set of keys for work:
 - F2 – save program.
 - F9 – check errors.
 - Ctrl + F9 – execute the program.
 - Alt + F5 – see results.
- Send solutions for testing to the system DL.GSU.BY.
- Use the test assignment (watch the input and output, on which the program gives the wrong answer).
- Search and correct errors in the program by comparing the correct output to the output of the program.

Note that by this time, after completing the first two parts of the training (“Learning to think”, “Learning words”), pupils already know how to turn on / off the computer, log into the network, launch the desired program by clicking on the desktop shortcut, to log in with a personal account in the DL system and understand the numerical task. In connection with the automatic differentiated learning, the system itself issues to each pupil a task on which he stopped.

At the same time, it is very important to preserve the motivation of children to practice, so the tasks should be diverse in form, interesting in content and provide differentiated learning, so that each pupil could find a comfortable mode of assignment.

The tasks themselves are lined up in a tree-like form, where the pupil can either proceed to the next task if he has coped with the current one, or to the task system that teaches how to complete the current task.

The words	Program
Program	program p1;
var	var
longint;	s : longint;
begin	begin
readln	readln(s);
writeln	writeln(s);
end.	end.

Fig. 4. Words and the program «Input-output numbers».

The standard view of the tree of learning the source code for solving a specific problem is presented in Fig.5.

Fig. 7–14 present the corresponding tasks for teaching the solution of the problem presented in Fig. 6.

The first task that is offered to a pupil who does not get to write the required program is the task (Fig. 7), offering the pupil to do the work instead of the program. That is, to indicate what the program will display if the specified numbers are entered at its input. This ensures that the pupil understands what the program should do. In case the child cannot cope with this task, there is a task in which the numbers necessary for input are displayed on the keyboard. This is especially convenient for self-study. In the classroom, an understanding of the condition may also arise from discussion of a problem with a teacher or another pupil designated by the teacher.

After the pupil has completed the task “Manual solution”, presented in Fig. 7, he is invited to type Pascal-program (Fig. 9). In this task, the algorithm for solving the problem is presented on the left, and on the right, you need to enter a program corresponding to this algorithm. And from this point on, differentiated learning begins to work most intensively.

There are children who just have to look at this picture to exclaim “I understand”, to press this button and return to the original task (Fig. 6), write the necessary program correctly, check, send it for testing and passing, go to the next task.

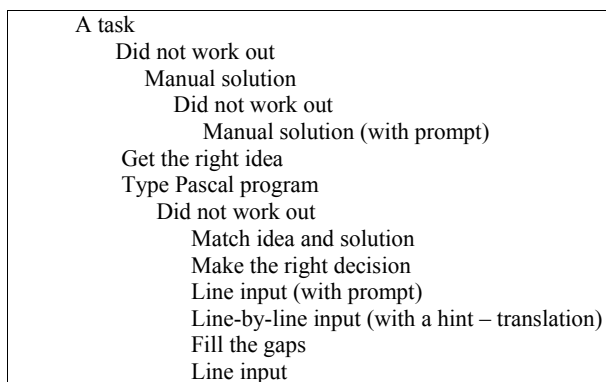


Fig. 5. Standard learning task tree.

Example of data entry:
3
5
Example of data output
3+1=4
4+5=9

Fig. 6. Example task to learn.

There are children who, exclaiming «I understand», still cannot write the program correctly and therefore have to return to the task presented in Fig. 9 again and write the program line by line on the model. If they can do this without a single error, they will return to the task in Fig. 6. If they make a mistake in at least one symbol or immediately press the “I do not know” button, they will get on the task “Create the right idea”, shown in Fig. 8. Here it is required using the permutation of the lines (by clicking on the two lines that need to be swapped) to make the algorithm presented in the previous task. For

Enter three numbers
Determine the difference between the first and second number, add the third number.
Print all operations.

Fig. 7. Manual solution.

Fig. 8. Get the right idea.

Left side: explanatory lines describing what needs to be done (originally in Russian)	Right side: the lines from the Turbo Pascal solution that perform the necessary actions
Program olymp	
variable	
s1, s2, s3, s4 : number	
begin	
read s1	
read s2	
read s3	
s4=s1-s2+s3	
write s1, '-', s2, '+', s3, '=', s4	
end	

Fig. 9. Type Pascal program.

those who cannot cope with this task correctly, there is a button “Show correct answers”, by clicking on which the lines are arranged in the correct order, however, the possibility of moving to the next task is blocked, and for the pupil it is required to click on the “Disable hint «and perform the task yourself.

Such a process can be repeated several times, until the pupil memorizes correctly the sequence of actions in the algorithm. After that, he is offered the task “To Match the Idea and the Solution”, presented in Fig.10. On the left there is the algorithm already mastered by the pupil in the past assignment, and on the right – the lines of the corresponding Pascal program, located in random order. The pupil must permute the lines of the program to turn it into the correct one. After that, the pupil can click “I understand” again and take the decision of the main task (Fig. 6.). Some pupils do this, they succeed, and they move on.

Some pupils fail, and they come back to this point. Some are not distracted and continue to perform learning tasks consistently. In order not to repeat – at any moment any pupil can press the “I understand” button, to try to take the completed task and, if it is decided wrong, to go back.

So, then the pupil is invited to the task “Make a decision”, presented in Fig. 11. In this task, permutation of the lines is required to make a program. In case of failure, you can use the buttons “Show correct answers” / “Disable hint”. In the case of this task, the transition to the “Line input” task, shown in Fig. 12, takes place.

An important feature of this task is the color support. That is, if the child is typing correctly, then everything that he is typing glows green, and when the first erroneous character is typed, the entire line of the set becomes red to signal the pupil about making a mistake at the exact moment when he made it. If the pupil succeeds to dial a few characters, “by inertia”, without noticing the mistake, it is enough to delete them one by one from the end until the line turns green again.

I would like to emphasize that we work with all the pupils who come – with completely different skills. And such a task helps the teacher to cope with “problem” pupils.

программа олимп;	s4:=s3+s2;
объявление переменных	readln(s2);
начало	program олимп;
читать (s1);	s3:=s1+1;
читать (s2);	end.
s3:=s1+1;	writeln(s1, '+1=', s3);
s4:=s3+s2;	var
писать (s1, '+1=', s3);	s1, s2, s3, s4 : longint;
писать (s3, '+', s2, '=', s4);	begin
конец.	readln(s1);
	writeln(s3, '+', s2, '=', s4);

Fig. 10. Match idea and solution.

```

s4:=s3+s2;
readln(s2);
program olymp;
s3:=s1+1;
end.
writeln(s1, '+1=', s3);
var
s1, s2, s3, s4 : longint;
begin
readln(s1);
writeln(s3, '+', s2, '=', s4);

```

Fig. 11 Make a solution.

```

program olymp;
var
s1, s2, s3, s4 : longint;
begin
readln(s1);
readln(s2);
s3:=s1+1;
s4:=s3+s2;
writeln(s1, '+1=', s3);
writeln(s3, '+', s2, '=', s4);
end.

```

Fig. 12. Line input.

After all, the standard alternative is that the teacher should look for (himself or with the help of other pupils) every pupil who made a mistake in the case where the pupil cannot cope with it himself.

Moreover, there are other assignments tasks presented in Fig. 13–14, in which you also need to perform keyboard input of the program, which also provides color error prompts.

In task 13 there are tips in Russian, in task 14 – it is emphasized what exactly has changed in this program in relation to the previous ones. The last task in this series (not shown in the figures) is a line-by-line entry of a program with a color prompt of errors “from memory”, that is, in the absence of prompting texts. In the case of this task, the pupil is assigned to the task shown in Fig. 9, which checks whether the pupil is able to type the program in the absence of color error prompts. This ensures that the program is memorized by a pupil with any level of prior training. At the same time, more prepared children undergo training faster; less prepared children spend exactly as much time as each of them needs personally to learn how to solve this problem.

```

программа олимп;
[ ]
переменная
[ ]
s1,s2,s3,s4 : число;
[ ]
начало
[ ]
читать (s1) ;
[ ]
читать (s2) ;
[ ]
s3:=s1+1;
[ ]
s4:=s3+s2;
[ ]
писать (s1, '+1=', s3) ;
[ ]
писать (s3, '+', s2, '=', s4) ;
[ ]
конец.
[ ]
    
```

Fig. 13. Line Input 2.

```

program olymp;
var
  s1,s2,[ ] : longint;
begin
  readln(s1);
  readln(s2);
  [ ] ;
  [ ] ;
  writeln([ ] );
  writeln([ ] );
end.
    
```

Fig. 14. Fill the gaps.

Differentiated Training

Under the differentiated training, the authors understand the creation of such a system of studies, when each pupil receives tasks that are feasible for him in complexity, and at the same time leading (albeit at different speeds) to the overall final goal.

Differentiation of training is provided by a rich choice of tasks for working at the table and a computer, automatic issuance of tasks on a computer depending on the results of the previous task, as well as a large set of task packages supporting the multiplicity of entry points to training.

Conclusion

In this article the author's approach to the beginning of learning programming in elementary school is considered. Many tasks of varying complexity, both on a computer and for working at a table, are of particular importance. Their proper use provides a differentiated approach to pupils with different training and motivation to practice. It is also important for the authors that, as practice shows, this training system is easily scaled and can be used even by teachers and parents, who were originally far from programming. At first, co-education may occur. But in the end, the children "gaining speed", have the ability to successfully engage in their own study.

References

- Dolinsky, M. (2013). An approach to teach introductory-level computer programming. *Olympiads in Informatics*, 7, 14–22.
- Dolinsky, M. (2014). Technology for the development of thinking of preschool children and primary school children. *Olympiads in Informatics*, 8, 63–68.
- Dolinsky M. (2016). Gomel training school for Olympiads in Informatics. *Olympiads in Informatics*, 10, 237–247.
- Dolinsky, M. (2017). A New Generation Distance Learning System for Programming and Olympiads in Informatics
- Dolinsky, M., Dolinskaya, M. (2018). How to Start Teaching Programming at Primary School? *Olympiads in Informatics*, 12, 13–24.
- Performance Statistics of Gomel pupils at international and national olympiads in informatics since 1997 up to 2018. (In Russian): <http://dl.gsu.by/olymp/result.asp>



M. Dolinsky is a lecturer in Gomel State University "Fr. Scoryna" from 1993. Since 1999 he is leading developer of the educational site of the University (dl.gsu.by). Since 1997 he is heading preparation of the scholars in Gomel to participate in programming contests and Olympiad in informatics. He was a deputy leader of the team of Belarus for IOI'2006, IOI'2007, IOI'2008 and IOI'2009. His PhD is devoted to the tools for digital system design. His current research is in teaching Computer Science and Mathematics from early age.



M. Dolinskaya is student in Gomel State University "Fr. Scoryna" from 2005 then graduate student from 2017. Since 2006 she is one of developer of the educational site dl.gsu.by as well as teacher of pupils from first grade. Her current research is in teaching programming from early age.

On Implicit Means of Algorithmic Problem Solving

David GINAT

*Tel-Aviv University, Science Education Department
Ramat Aviv, Tel-Aviv, Israel 69978
e-mail: ginat@post.tau.ac.il*

Abstract. Students of challenging algorithmics learn and utilize a variety of problem solving tools. The primary tools are data structures, generic algorithms, and algorithm design techniques. However, for solving challenging problems, one needs more than that. Many creative solutions involve implicit notions, whose creative employments yield elegant, concise, and efficient solutions. We elaborate on such notions and advocate their relevance as valuable means in one’s problem solving toolbox. We display our experience with students who lacked awareness of these notions, and illustrate the relevant role of three such notions – the notions of “candidate”, “complement”, and “invariance”.

Keywords: algorithmic problem solving.

1. Introduction

The algorithmic problem solving toolbox includes a variety of generic design patterns and techniques. The generic design patterns, or schemes, include algorithms such as sorting, searching, and graph algorithms. The design techniques include problem solving strategies such as top-down design, divide and conquer, the greedy strategy, dynamic programming, reduction, and more (e.g., Cormen *et al.*, 1990). Students learn and employ these patterns and techniques, together with a variety of data structures, from the very basic courses to the more advanced algorithms courses. Yet, more can be offered to learners, in particular those of challenging algorithmics.

For example, the very basic pattern of max computation involves the notion of “candidate”. The computation involves running through a list of values, while keeping the largest value read so far as the current candidate for the answer. While the design pattern is very basic, the notion of “candidate” is a notion that may be used in advanced, creative ways in challenging algorithmics. One challenging task in which it is creatively applied is the task of seeking majority (i.e., a value that appears a majority

number of times) in a very very long list of values (Boyer and Moor, 1991). The input cannot all be kept in memory (by elements or range), but may be read more than once. Reading is expensive. The challenge is to devise a solution that involves reading the input list as fewer times as possible.

An efficient, elegant solution is based on the following insightful declarative observation: *if the value v is in majority in a list of N values, and we delete v and a different value u , then v is still in majority in the remaining list of $N-2$ values*. This observation yields an appealing use of the notion of candidate. The candidate will be the element that will remain after deleting pairs of different values, and will be checked for being majority. The computation will involve two passes over the input: a first pass for deleting such pairs and leaving a sole candidate for majority (if there is one), and a second pass for checking whether this candidate is indeed a majority (Boyer and Moor, 1991; Ginat, 2002).

Creative utilizations of the notion of “candidate” appear in a variety of additional task solutions. And there are more soft notions that are repeatedly employed in creative ways in algorithmic solutions. Additional notions are: “complement”, “symmetry”, “parity”, “invariance”, “the pigeon-hole principle”, and more. Although these notions play a significant role in algorithmic solutions, they are not underlined or explicitly mentioned in textbooks and teaching materials. Perhaps this is due to tutors’ assumptions that the implicit utilization of these notions may be sufficient for acquiring them and invoking them. In our experience, this is not always the case, even with talented students of challenging algorithmics. Students may benefit from explicit indication and practice of these notions.

In what follows we describe our ‘notion-invocations’ experience with the students of our advanced OI stages in the last few years. We display several tasks whose solutions require invocations of such notions, reveal student difficulties, and present elegant employments of these notions. In the last, Discussion section we discuss our findings, and advocate elaboration of these notions as important means of algorithmic problem solving.

2. Implicit Problem Solving Notions

In this section we display three notions that are useful in solving algorithmic tasks. The first notion – candidate – is particularly relevant in algorithmics. The other two notions – complement and invariance – are relevant in both algorithmics and mathematics.

All these notions are related to a declarative point of view, which involves a “what” characteristic perspective. This point of view precedes the operational point of view, which is related to “how”, of the computational actions. While novices often ‘get by’ without the declarative perspective, this perspective becomes more and more important as the algorithmic challenge increases.

The Notion of Candidate

The example in the Introduction displayed a creative utilization of “candidate”, based on a mathematical observation. The innovative feature in this utilization involved the decomposition of the computation into two sub-tasks – one of seeking a sole candidate, and another of checking whether this candidate is indeed the desired element. This utilization of the notion of candidate appears in additional tasks. One such task is the Celebrity problem (Manber, 1986), in which a person who knows nobody, but is known to all, is sought.

Utilizations of the notion of “candidate” may have a variety of forms. Another form appears in the solution to the Widest Inversion problem (Ginat, 2011), in which the longest distance between two unordered numbers is sought. The efficient solution of this task is based on pre-processing in which lists of candidates for the left-end and the right-end are found, and then elegantly processed, based on their characteristics.

The following task displays an additional, creative utilization of the notion of candidate, this time in an “as if” manner (Ginat, 2010). In our experience, a non-negligible amount of talented students did not turn to this perspective, but to a naive utilization of a candidate.

Longest Plateau. We define a *plateau* as a sequence of integers in which the difference between every two (not necessarily adjacent) is at most 1. Thus, 4 3 4 3 3 is a plateau, whereas 4 3 4 3 2 is not a plateau. Given a list of N integers, output the length of the longest plateau.

Example: For the input 2 3 3 4 3 5 5 4 3 3 2 3 2 2 1 3 the output should be 6, due to the sub-sequence 3 3 2 3 2 2.

Notice that a sub-sequence (part) of a plateau is also a plateau, though it is a plateau that may be extended to a longer one.

At first glance this task may seem boring and straightforward. Yet, this was not the case for quite a few of our students. It was particularly so when we asked for an extension, in which the definition of a plateau allowed a difference of K , greater than 1 between every two elements of a plateau.

The challenge in the task stems from the possibility of plateaus’ overlap. For example, in the sequence 2 3 3 4 3 the plateaus 2 3 3, and 3 3 4 3 overlap, as they share the sub-sequence 3 3.

Many students offered a cumbersome on-the-fly solution using a single candidate for the current plateau. The underlying idea in their solution was to accumulate the current plateau, while handling a series of history considerations of sub-sequences’ overlaps. They kept the information of the latest overlap history and manipulated it when an overlap ended. While this idea may be suitable, it yielded cumbersome and erroneous implementations. In addition, it is not extendible for more general plateaus.

One may do better, and simpler if one abstracts the view of the task, by looking at each input value “as if” it belongs to two “active” plateaus concurrently – one in which this input value is the lower value, and one in which it is the higher value. Thus, at any

given time, the current element belongs to **two concurrent** candidate plateaus. When one of the candidates ends, its length is compared to the longest plateau so-far.

Each input value v may be viewed as contributing a “block” to its *lower candidate plateau*, and a “block” to its *upper candidate plateau*. For example, upon looking at the sub-sequence 2 3 3 4, starting at the value 2, this value contributes a block to its lower candidate-plateau $p^{1,2}$, and a block to its upper candidate-plateau $p^{2,3}$. The next value, 3, contributes a block to its smaller candidate-plateau $p^{2,3}$ (which was just the upper candidate-plateau of the 2) and a block to its upper candidate-plateau $p^{3,4}$.

Based on the above abstract view, we may devise the following *candidate-plateau scheme*. A new candidate-plateau begins when the next input value contributes a block that does not extend a current candidate-plateau. A candidate-plateau is extended by a block, when the next input value contributes a block to it. A candidate-plateau ends when the next input value does not extend it. Fig. 1 below illustrates visually the scheme with the input of the problem statement.

This concise scheme annuls the need of keeping history details, and requires that we remember at any given time only the length of the longest plateau ended so far, and the two current candidate-plateaus. The scheme’s underlying perspective is specified declaratively, by relating two “as if” candidate plateaus to every input element.

The Notion of Complement

The notion of complement is a powerful notion in both mathematics and computer science. Its underlying principle is that sometimes it may be more beneficial to look at “the whole” minus the given elements rather than at the given elements. We illustrate it with the following task (which we learnt from IOI colleagues).

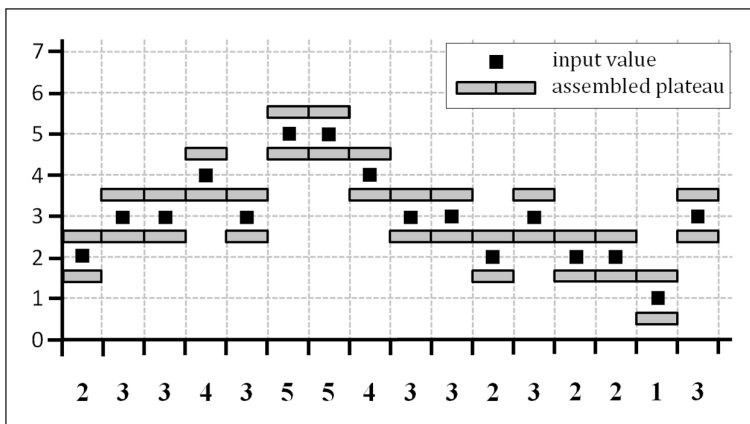


Fig 1. A view of the candidate-plateaus. The input is the sequence of values written under the X-axis. The small black boxes represent the input values, and the grey sequences represent the candidate-plateaus that were assembled from left to right.

Safari. In a very large safari, with N animals, some animals get along with one another (and can stay together), and some do not. For every animal, an indication is provided about all the animals with which it gets along. The number of animals in the safari is a multiple of 3. It is known that exactly $2/3$ of the animals all get along with one another. Each of the other animals gets along with some subset of the N animals. The safari management wants to select exactly $1/3$ of the N animals to be put together. Given for each animal the animals with which it gets along, output $N/3$ animals that may be chosen to be put together.

A non-negligible amount of students struggled with this task. All of them represented the task with an undirected graph, and attempted various ways of processing the edges of the specified clique of $2N/3$ animals. Unfortunately, their attempts were either erroneous or cumbersome and inefficient. The suitable approach is not to process the given graph edges, but rather the missing edges, of the complement graph.

An edge in the complement graph connects two graph nodes (animals), such that one of them or both of them are not in the specified clique of $2N/3$ animals. Thus, if we **remove** the two **nodes** of a missing edge, and all their incident edges, then we remove **at most** one of the $2N/3$ -clique nodes. By repeating this process again and again, we remove at most $N/3$ animals from that clique. Therefore, when this process ends, the remaining graph will be a clique of at least $N/3$ animals. If it is exactly of that size, then all its animals will be displayed as output; and if it is larger, then any $N/3$ animals taken from this remaining graph will be a suitable output.

All in all, the key point, which paved the way for the surprisingly simple solution, was the utilization of the complement graph of the given input. Capitalizations on complement graphs appear in a variety of graph-based solutions. And the notion of complement appears in many other representations.

The Notion of Invariance

The notion of invariance involves assertions that are repeatedly kept during an iterative or distributed process. It is an essential means in mathematics and computer science. In both domains it is used for arguing and proving characteristics of sets and processes. In computer science it is a fundamental means in the domain of logic of programs, for describing loop executions and distributed-system characteristics. Dijkstra and others (e.g., Gries, 1981; Dijkstra, 1989) argue that computer-program design should go hand-in-hand with arguing its correctness, and the design of loops should be based on devising their underlying invariants. However, the notion of invariance is not advocated in most programming and algorithms textbooks, and its essential role is overlooked. We present below two short examples that demonstrate its importance.

Diplomats' Separation. N diplomats enter a large conference hall. Each of the diplomats has **at most** 3 rivals among the others. Rivalry is symmetric. The conference organizers were asked to separate the group of the N diplomats into two sub-groups – A and B – such that each diplomat will be with **at most** one rival in her group. Given

the rivalries of each diplomat, output a separation into two suitable sub-groups, or notify that it is impossible to obtain such separation.

The task is not a difficult task. Yet, some students approached it in a way that was intuitive for them, but not that successful. They offered the following scheme. Start with all the diplomats in sub-group A, and check for each diplomat whether there is more than one rival in her sub-group. If this is the case, then leave one rival in her sub-group, and **transfer the other(s)** to the other sub-group. Repeat this process, with sub-groups A and B, until the desired condition is met.

Although this intuitive process seems suitable, it is questionable, as a diplomat may be transferred back and forth between the sub-groups and it is not clear whether successful termination is guaranteed.

A small modification of the above scheme, supported by invariance argumentation, makes a big difference. When a diplomat is found to be in a sub-group with more than one rival – **transfer that diplomat** to the other group, rather than her rivals. This idea is supported by the following invariant assertion:

Each transfer of a diplomat reduces the total number of rival-diplomats inside the sub-group.

The assertion involves a *measure* (specified in its latter part). It is an *invariant assertion*, since it asserts a characteristic that is kept after **every** diplomat transfer. It implies that the suggested scheme **improves** the situation with every transfer. Even if a particular diplomat is transferred more than once, this iterative process is guaranteed to end, with a suitable separation of the diplomats into two sub-groups, since the decreasing number of rival-diplomats in the sub-groups cannot go below 0. The utilization of invariance and a measure that implies successful termination enfolds the fundamental computer science elements of safety and liveness (Alpern and Schnieder, 1985).

• • •

One sub-domain in which invariance is fundamental is that of mathematical games. Games are present in both mathematics and algorithmics. Many games are two-player games, in which a winning strategy for one of the players is required. The strategy may be specified with an algorithm, but the underlying characteristic that yields and justifies the algorithm is expressed with an invariant. For example, the first task in IOI'96 was a game with a line of $2N$ numbers, such that each player takes, in her turn, a number from one of its ends. The winning strategy was based on an elegant invariant: *after every move of the first player, the numbers in the two ends of the line are in locations that were originally of the same parity*. The following task also involves a game invariant.

DVD Game. Given a line of $2N$ cells, $N > 100$, two players play the following game. Each player, in her turn, writes a “D” or a “V” in an empty (not yet used) cell. The first player who completes the sequence “DVD” wins the game. (Players may use one another’s written letters in forming the desired sequence.) Devise a strategy for winning the game.

As in the previous task, although the challenge here is rather limited, quite a few of the students offered incomplete or erroneous solutions. Most of them noticed that the winner should be the second player (the one that does not start), and should create in her first two moves a “trap”, in the form of D__D. The player who will be forced to write a letter between the two D’s will lose the game.

At this point some of the students got confused because of the possibility of additional traps created during the game. They were unsure about the winner’s responses for the opponent’s moves. Their common strategy was to imitate the last opponent move, by writing the same letter that she just wrote, next to her letter. This strategy involved all kinds of cumbersome conditions for avoiding traps. Some were incorrect.

The one thing that many students missed was a simple invariant on which to capitalize:

*After each move of the first player, there is at least one sequence of empty cells of an **odd** length.*

Thus, after creating the trap in the first two moves, the winner has a very simple strategy: if it is possible to win the game in the next move, then do so; otherwise, write a “V” in the the middle of an odd-length sequence of empty cells.

It is simple to justify termination of the game with the second player’s win. The key point was to recognize and capitalize on the simple invariant. The students who offered the cumbersome, sometimes erroneous solutions indicated that they focused on an immediate reply to the opponent, next to her move, rather than on a global characteristic of the game line. This direction was heuristic, and lacked rigor.

3. Discussion

Chapters of algorithms textbooks are usually designed according to programming constructs, data structures, design patterns, and design techniques. These elements are the primary tools, or means for algorithmic problem solving. But they should not be presented as the only ones. Algorithmic problem solving involves a collection of implicit notions, which may be considered as tools, since they are repeatedly utilized in various ways, particularly in challenging algorithmics.

In many cases the employment of these notions is essential. They pave the way to a desired solution prior to utilization of the primary tools. They belong to the task analysis stage that should progress hand-in-hand with a solution design. While the characteristic of the primary tools is primarily operative, the nature of these notions is declarative. They direct the design and justify its outcome.

Competent problem solvers are acquainted with these notions, and possibly invoke and employ them without explicitly denoting them. They assimilate these notions upon learning and practicing implicit utilizations in a variety of task solutions. But not all students grasp these implicit notions. Explicit elaboration may considerably help, particularly for developing awareness of these notions.

In the previous section we presented our experience with talented students who did not turn to these notions and provided unsuitable solutions. One may say that at least some of these notions (such as the notion of complement) are common knowledge and should not be explicitly underlined. We believe that explicit indication and practice may lead students to seek utilizations of such notions. This is our experience with OI students. Repeated elaborations of these notions enhanced flexibility and abstraction upon approaching task solutions – flexibility in the sense of creative utilizations of familiar means; and abstraction in the sense of conceptual exploration (which yielded, for example, the “as if” perspective in the Longest Plateau example).

Such notions also upgrade one’s discipline in the process of problem solving. Awareness may lead one’s attempts to employ a declarative point of view prior to an operative implementation. The declarative perspective may yield further insight, which will assist in considering different alternatives of progress. In addition, it may strengthen one’s conviction and rigorous argumentation about her solution.

We illustrated creative employments of the notions of candidate, complement, and invariance. The notion of candidate is unique to algorithmics, as it is explicitly tied to progression during a computation process. The notions of complement and invariance are relevant also in mathematics. They may be regarded as resources in the problem solving model of Schoenfeld (1992), in the sense of means with which one should be acquainted for recognizing, specifying, and justifying characteristics. In algorithmics, they are also relevant for devising concise and efficient computations.

More illustrations of these notions will enrich problem solvers’ toolboxes. Tutors aware of these notions may embed and underline their explicit utilizations during the teaching of design patterns and design techniques. Repeated embedment and elaboration will raise students’ computational thinking competence.

Acknowledgement

I thank my colleagues Hanit Galili and Sharon Zuhovitzky from our Israel OI team for the Safari task story, and its student solutions’ summary.

References

- Alpern, B., Schneider, F.B. (1985). Defining Liveness. *Information Processing Letters*, 21, 181–185.
- Boyer, R.S., Moor, J.S. (1991). A fast majority vote algorithm. In: *Automated Reasoning: Essays in Honor of Woody Bledsoe*. Automated Reasoning Series. Kluwer, 105–117. (The algorithm was invented in 1980.)
- Cormen, T.H., Leiserson, C.E., Rivest, R.L. (1990). *Introduction to Algorithms*, MIT Press.
- Dijkstra, E.W. et al. (1989). A debate on teaching computing science. *Communications of the ACM*, 32(12), 1397–1414.
- Ginat, D. (2002). On varying perspectives of problem decomposition. In: *Proc of the 33rd ACM Computer Science Education Symposium – SIGCSE*. ACM Press, 331–335.
- Ginat, D. (2010). The baffling CS notions of “as-if” and “don’t care”. In: *Proc of the 41st ACM Computer Science Education Symposium – SIGCSE*. ACM Press, 385–389.

- Ginat, D. (2011). Algorithmic problem solving and novel associations. *Olympiads in Informatics*, 5, 3–11.
- Gries, D. (1981). *The Science of Programming*. Springer.
- Manber, U. (1986). *Introduction to Algorithms: a Creative Approach*. Addison Wesley.
- Schoenfeld, A.H. (1992). Learning to think mathematically: Problem solving, metacognition, and sense making in mathematics. In: Grouws D.A. (Ed.), *Handbook of Research on Mathematics Teaching and Learning*. 334–370.



D. Ginat – heads the Israel IOI project since 1997. He is the head of the Computer Science Group in the Science Education Department at Tel-Aviv University. His PhD is in the Computer Science domains of distributed algorithms and amortized analysis. His current research is in Computer Science and Mathematics Education, with particular focus on various aspects of problem solving.

Multidisciplinary, Multilingual, Multilevel and Multipurpose Usage of GeoGebra Software in Education

Metodija JANCHESKI¹, Sofija JANCHESKA²

¹*University Ss. Cyril and Methodius, Faculty of Computer Science and Engineering
Rudzer Boshkovikj street, 16, 1000 Skopje, Macedonia*

²*New York University Abu Dhabi, Faculty of Computer Science
Saadiyat Island, Abu Dhabi, United Arab Emirates*

e-mail: metodija.jancheski@finki.ukim.mk, sofija.jancheska@nyu.edu

Abstract. The technical characteristics of GeoGebra have been examined, its various views and tools have been described, with special emphasis on interactive tools. The use of GeoGebra in education has been depicted, mainly in mathematics. We emphasized the importance of sharing GeoGebra digital learning materials within the GeoGebra community, which is present and active in more than 190 countries now and is growing astonishingly fast. In this context, various forms of organizing GeoGebra digital learning materials are reviewed: GeoGebra Materials, GeoGebra Wiki, GeoGebra Tube, GeoGebra Books and GeoGebra Exercises.

The author demonstrates the universal and effective application of the GeoGebra software in four dimensions. First, GeoGebra can be applied in a variety of disciplines, primarily in mathematics and computer science, and in subjects in the field of natural and technical sciences. Second, the use of GeoGebra also covers a wide range of applications, from primary education to higher education. Third, GeoGebra enables the creation of multilingual digital learning content. Finally, various variations of animations and simulations with different weights can be made in GeoGebra, which can enhance the individualization of teaching within multiple levels of education and enable conditions for programmed instruction.

A detailed overview of three conducted researches and the obtained results are provided. The researches include filling in questionnaires by teachers and students, as well as performing teaching lessons in Mathematics and Physics in two secondary schools with students divided into experimental groups (where GeoGebra animations and simulations were applied) and control groups (where the classes were held in classical, traditional way, without the use of educational software).

According to our researches and observations, free and open-source software like GeoGebra is a great opportunity that should be used to the fullest extent in our education especially in the conditions of obvious need for educational software and digital educational materials, corresponding to our education curricula. In the concluding observations, we give concrete conclusions, suggestions and recommendations for implementing GeoGebra in the educational system.

Keywords: GeoGebra, application of GeoGebra in education, free and open educational software.

1. Introduction

GeoGebra is a professionally developed, freely accessible, interactive and dynamic educational mathematical open source software for teaching and learning mathematics at all levels of education, from elementary school to university (GeoGebra, 2019), (Chrysanthou, 2008), (Kllogjeri and Kllogjeri, 2014). It is written in Java programming language, and its web application in HTML5. GeoGebra derives its name from the fact that it unites mathematical disciplines algebra and geometry (Mukiri, 2016). The creator of GeoGebra is Markus Hohenwarter, who first introduced this software in his master's thesis in the field of mathematics and computer science, defended at the University of Salzburg, Austria in 2002 (Hohenwarter and Lavicza, 2010), (Mukiri, 2016). Since 2006, the GeoGebra project has enjoyed the support of the Ministry of Education of Austria, which provides access to GeoGebra for mathematics education in schools and universities around the globe.

Since July 2006, the development of GeoGebra continues in the United States at the University of Florida Atlantic, where it is actively used in regular classes. (Mukiri, 2016)

GeoGebra program is a free program (GNU license) used worldwide. It is available in more than 64 world languages (Hohenwarter and Lavicza, 2010) which allows the use of local language software and multicultural learning environments (Chrysanthou, 2008). In many educational systems around the world, GeoGebra is integrated into textbooks and various projects.

GeoGebra has received more prestigious international awards for the best educational software, including: Archimedes 2016, MERLOT Classics Award 2013, NTLC Award 2010, Tech Award 2009, BETT Award 2009, SourceForge.net Community Choice Awards 2008, AECT Distinguished Development Award 2008, Learnie Award 2006, eTwinning Award 2006, Trophées du Libre 2005, Comenius 2004, Learnie Award 2005, Digita 2004, Learnie Award 2003, EASA 2002. (GeoGebra, 2019), (Hohenwarter and Lavicza, 2010).

The GeoGebra community exists in almost every country in the world. It is constantly evolving and has millions of users.

2. Technical Characteristics of GeoGebra

2.1. Basic Information

GeoGebra is designed to combine the properties of dynamic geometric software (for example, Cabri Geometry, Geometer's Sketchpad, Euclides, Geonext, Descartes, Cinderella, EucliDraw, and others) and a computer algebra system (for example, Derive, Maple) in a single, integrated and easy-to-use system for teaching and learning mathematics (Hohenwarter & Preiner, 2007b) (Hohenwarter and Lavicza, 2010). GeoGebra has the ability to bridge the differences that exist between mathematical

disciplines, especially between geometry, algebra, and mathematical analysis (Kllogjeri and Kllogjeri, 2014).

The latest stable version of GeoGebra is the version 6.0.523.0 from January 31, 2019. It can be used on several operating systems: Windows, MacOS, Debian, Ubuntu, Red Hat, Linux, openSuse, Android, iOS, but also as a web application.

Although though GeoGebra possesses powerful features, it does not have large system requirements. It can be installed on all operating systems and is really straightforward. GeoGebra’s virtual tools can be easily used, and the GeoGebra environment can be very cozy and attractive because the work with it resembles playing a computer game (Kllogjeri and Kllogjeri, 2014).

GeoGebra’s graphics view is of high quality, and the drawings made by GeoGebra can be easily transferred to other presentations and programs (for example, Latex). GeoGebra can be used to create interactive tutorials, animations and website simulations (GeoGebra, 2019). GeoGebra’s user interface is flexible and can be tailored to the needs of students. According to Hohenwarter and Preiner (2007), GeoGebra is a friendly software that can be used intuitively and does not require advanced skills. (Kllogjeri and Kllogjeri, 2014)

2.2. Description of GeoGebra views and Toolbar

GeoGebra’s interface includes: a menu, a toolbar, Algebra view (or window), Graphics view (window), Spreadsheet view (window), and Input bar (Fig. 1).

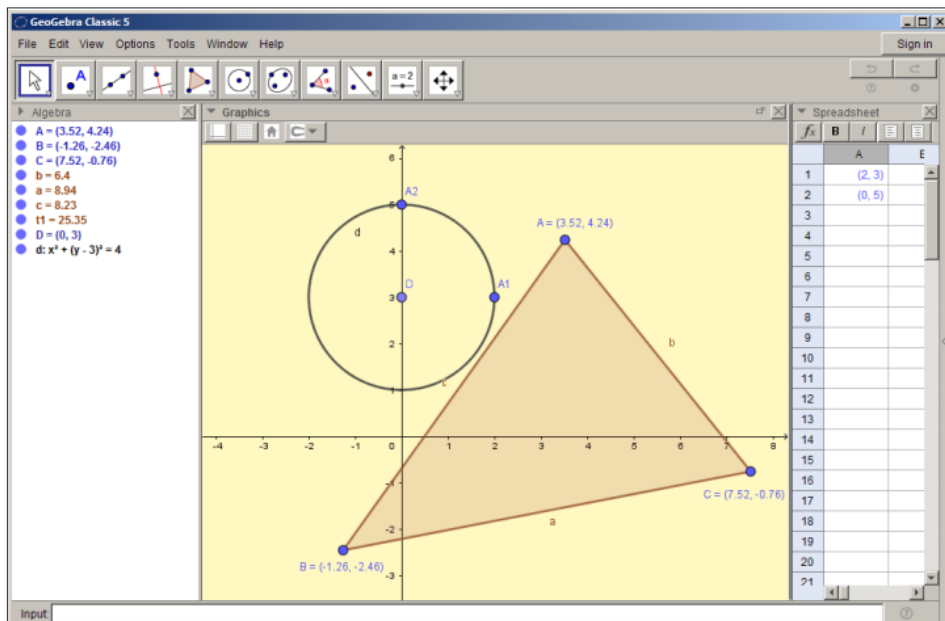


Fig. 1. Three GeoGebra views (windows).

By using the Input bar and the Enter key, the following data can be directly entered in the Algebra view of GeoGebra: constant values and parameters, coordinates of points and vectors, complex numbers, mathematical expressions (arithmetic, algebraic, trigonometric, analytical), equations and inequalities (algebraic, transcendent, differential, integral), systems of equations and inequalities, commands and functions with one or more variables (linear, square, cubic, exponential, logarithmic, and others). A more detailed perspective of the Algebra view is the Construction protocol. It describes all steps of the interaction between the user and the computer.

Using the virtual tools from the GeoGebra toolbar, a variety of geometric objects can be constructed in the Graphics view, including: points, vectors, angles, lines, line segments, series of line segments, polygons, graphs of functions, curves, conic sections, and other more complex objects. In this view, the coordinate axes and the coordinate grid can be turned on/off, and text can also be entered.

Using the spreadsheet, numerical data can be entered in the worksheet. They can later be used for creating lists of values, lists of points and matrices. Also, they can be applied for statistical data processing, hypotheses testing for particular distributions, etc.

We concluded that there are three ways of displaying the objects: Algebra view, Graphics view and Spreadsheet view. If we create (construct) a geometric object in the Graphics view through an appropriate geometric tool, then the corresponding algebraic representation (mathematical expression, command and function) of that object will automatically be created in the Algebra view. On the other hand, after each direct input (via a keyboard) of a mathematical expression, command, or function in the Input bar, an appropriate geometric object is automatically created in the Graphics view. Thus, the object (objects) in the Graphics view corresponds to the expression (expressions) in the Algebra view and vice versa. Any change (transformation) of an object in the Graphics view automatically causes an appropriate change of algebraic properties of that object, which leads to an appropriate alteration of the Algebra view content. The reverse is also true. So, all displays of an object are dynamically connected and, regardless of the way the object was created, any change in any of its views (displays) causes changes in other views (displays). Multiple representations of objects are one of the greatest advantages of GeoGebra.

2.3. Interactive Tools in GeoGebra

In order to include interactivity in the GeoGebra documents, we can use the following types of objects, included in the toolbox:

1. Check boxes; they represent graphical representations of '0' and '1'.
2. Input boxes; they have the same function as text input for scripts. The scripts are activated in the following three cases: when the text is changed in the Input box, when the Enter key is pressed or when the Input box is abandoned.
3. Buttons; clicking a certain button activates its corresponding script. Although scripts can also be activated by clicking on other objects, such as images, the use of buttons makes the applications more intuitive.

4. Dropdown lists; they are used to display the content of already created lists in the form of a drop-down menu. To create a drop-down list, the user needs to check the “Draw as drop-down list” field on the “Basics” tab of the “Object Properties” dialog box. It is activated from the context menu which is generated by right-clicking on the appropriate list (the list which is intended to be displayed in the form of a drop-down menu). Drop-down lists can also be created with appropriate commands.

The first three types of objects can be formed in two ways, through the appropriate tool from the toolbox or by entering the corresponding command in the Input bar.

3. International GeoGebra Cooperation and Open Educational Resources

3.1. International GeoGebra Cooperation

Over the past few decades, it has been shown that a group of enthusiasts can change people’s conventional thinking and patterns for development and innovation. The success of open source projects such as Linux, Firefox, Android, MOODLE and Wikipedia have shown that collaboration and sharing can produce valuable resources in different areas of life. Hohenwarter and Lavicza (Hohenwarter and Lavicza, 2010) emphasize the fact that the international GeoGebra community raised the awareness about the importance of GeoGebra.

GeoGebra is a fast growing community with over 50 million users from around 190 countries worldwide (Stephen Jull’s LinkedIn page, 2018). GeoGebra’s open source code encourages communication and collaboration among its users across the world. Users can contribute with their own creations or produce customized versions of existing worksheets. Questions and ideas about GeoGebra can be discussed in GeoGebra’s user forum (GeoGebra User Forum, 2019).

3.2. International GeoGebra Institute

At the end of 2007, the International GeoGebra Institute (IGI) was established to support GeoGebra community members and teachers who were starting to use GeoGebra by offering a community forum that will expand cooperation and interconnections. (Hohenwarter and Lavicza, 2010)

The International GeoGebra Institute is the main umbrella organization of local GeoGebra institutes founded by university teachers and researchers and teacher education institutions (Hohenwarter & Lavicza, 2007). Four main objectives of the International GeoGebra Institute are: trainings and support for teachers, preparation of teaching materials and software, research and coverage of less developed communities. The local GeoGebra institutes follow the goals of the international GeoGebra Institute, but the emphasis on their work depends on their local needs, interests and priorities. (Hohenwarter, Lavicza)

There are currently 132 GeoGebra Institutes in 58 countries in the world (GeoGebra Institutes, 2019).

3.3. *GeoGebra Materials*

GeoGebra's users (teachers, students, researchers, software developers and other enthusiasts) can create original interactive dynamic teaching materials (interactive GeoGebra websites, applets, constructions, and other learning resources) that can then be used and shared online via a platform called GeoGebra Materials. It contains over 170.000 materials and is constantly increasing on a daily basis. Users can also take an advantage of the software to solve open problems. (Stephen Jull's LinkedIn page, 2018)

The GeoGebraWiki pool (GeoGebra Wiki Pool, 2019) appeared shortly after the appearance of GeoGebra. In 2011, the successor to the GeoGebra Wiki Platform, also known as GeoGebraTube, appears, in line with the role it has, since it practically represented YouTube for free GeoGebra materials. The GeoGebraTube platform was renamed in 2016 to GeoGebra Materials. GeoGebra Materials can also be defined as an official cloud service and a repository of interactive resources for learning and teaching. This service has over million resources, over 40% of which are publicly shared as searchable materials – such as interactive worksheets, simulations, games and e-books created through the GeoGebraBook. In each book's chapter, worksheets can be stored. Through simple metadata, you can easily search for existing materials regarding different levels, themes, and languages.

GeoGebra materials can be exported in multiple formats, such as static images or animations. Issues of organizing and ensuring the quality of resources, as well as linking them to other open educational resources platforms, are still open.

3.4. *GeoGebra Exercises*

Interactive applets ("Mathlets") can be created in the framework of the GeoGebra Exercises project, which will generate random questions to suit students. They provide feedback to the students (visual or textual). The response checks are done with a combination of JavaScript and GeoGebra (including the GeoGebra symbolic algebraic system).

These interactive applets can then be imported as a SCORM package in a learning management system, for example MOODLE, and the results will be recorded. The level can range from primary (e.g, fractions) to advanced (for instance, mathematical analysis), and the goal is to fully cover the entire high school curriculum. (Hohenwarter&Lavicza)

First, GeoGebra can be applied in a variety of disciplines, primarily in mathematics and computer science, and in subjects in the field of natural and technical sciences. Second, the use of GeoGebra covers a wide range of applications, from primary education to higher education. Third, GeoGebra enables the creation of multilingual digital learning content. A wide range of animations and simulations with different difficulty levels can be made in GeoGebra, which can enhance the individualization of teaching within multiple levels of education and enable conditions for programmed instruction.

4. The Application of GeoGebra in Education

4.1. *GeoGebra and Education*

GeoGebra is created for educational purposes. It combines contents of the fields of geometry, algebra, statistics, analysis, and spreadsheets and graphics in a simple, easy-to-use package (GeoGebra, 2019), (Leggett, 2014). GeoGebra is a virtual experimental laboratory which can be used: as a place where hypotheses can be posed and tested; to monitor, verify and confirm scientific facts; to illustrate tasks and problems, their variations and sets of solutions; to analyze the properties of different geometric objects; to discover geometric points of points that satisfy certain conditions, to find important points of functions; to prove mathematical statements; to demonstrate various evidence of mathematical statements and the like.

We took particular care to ensure that the content and weight of digital educational materials included in different levels of education correspond to the age of students and can be adapted to the curricula of the respective subjects. It is important to note that where it was allowed, a part of the digital learning materials created for a particular subject, for example Mathematics for primary education, were used as a basis for the development of appropriate digital learning materials for Mathematics for secondary education, with carefully adjusted degree of complexity. The same was true in reverse, applying the appropriate simplification.

Also, almost any digital content can serve as a basis for creating other digital content. Namely, the created simulations include several parameters that can be manipulated by the student. The involvement of only one parameter means that the simulation includes an entire family of tasks, situations, and events corresponding to that parameter.

GeoGebra is equally popular among students and teachers. Students are attracted primarily to its dynamism and interactivity, but also to its ability to provide a visual and conceptual feedback. It has been acknowledged that GeoGebra encourages students to learn mathematics (Hohenwarter and Preiner, 2007), helps them to analyze various problems and to simulate physical phenomena through dynamic structures. It offers unlimited experimental opportunities to students, which leads to an improved content understanding and more effective learning. GeoGebra is a free software and students can use it not only at school, but also at home, while doing homeworks, practicing, revising the courses' material and preparing for future lessons. (Chrysanthou, 2008)

The most common application of GeoGebra is in the field of teaching and learning mathematics. It can simultaneously serve as a tool for demonstrating and visualizing formal mathematical knowledge, authoring, creating interactive learning materials and facilitating collaboration and communication. Except as a computer tool, many educators consider GeoGebra a conceptual, pedagogical, cognitive and transformational tool in teaching and learning mathematics. This highlights the universality of GeoGebra in teaching mathematics and educational reforms in the field of mathematics. (Chrysanthou, 2008), (Bu and Schoen, 2011)

The flexibility of GeoGebra is multifunctional, it can be used in teaching Mathematics at all levels of education and in a wide range, from simple to complex structures. It is suitable for students with varying degrees of ability and can be used to master teaching content in other subjects (Physics, Chemistry, Biology, Astronomy, etc.). The GeoGebra website considers GeoGebra a supporter of education in natural sciences, technology, engineering and mathematics (STEM) and innovation in teaching and learning around the world. According to Chrysanthou (Chrysanthou, 2008), GeoGebra promotes mathematical research and can enable effective application of constructive, cognitive and collaborative learning models in educational institutions at different levels. The extent of teachers who use GeoGebra is also wide, ranging from the preparation of teaching materials to making tests for students' knowledge assessment.

4.2. GeoGebra in Computer Science Education

Students often times have troubles when they are assigned to write a computer program. Throughout an extensive experience with students in the Computer Science field, we have detected that the main problems lie in their lack of understanding or misunderstanding about what has been assigned to them and how certain algorithms function. Mainly, students are given an initial problem and appropriate input data and they are asked to solve it using an effective algorithm. If students do not properly understand the question or the algorithm, they will not be able to cope with it in the correct way. Appropriately created and adapted animations, simulations and computer games portraying visual representation of algorithms can certainly contribute to students' better understanding about the assigned problems. GeoGebra does exactly this; its interactive nature helps students easily submerge into the secrets of computer programming and algorithms. One such example of a GeoGebra animation is illustrated in Fig. 2 where different sorting algorithms (Selection sort, Insertion sort, Merge sort, Bubble sort and Quick sort) are applied. By selecting any of the sorting algorithms, students can experiment with different sets of data input. This will not only show the correct solution (a sorted array), but it will show all its steps as well. Due to GeoGebra's visualization features, particularly the red and green colors applied to the numbers in this example, students can observe the actual comparison and predict the shifting of the numbers. Additionally, students can choose the next step button (step-by-step manner) or the animation button (all steps, one after another) with their own assigned pace.

Such an example solves the above-mentioned problem regarding the appropriate comprehension of the problem or algorithm. This problem solving approach will engage students from all levels, starting from young age to upper-class students, but will especially help the ones that have troubles with the school's material. GeoGebra will intellectually stimulate them to draw their own patterns about the work of certain algorithms and help them create a computer program with the correct solution of the problem. For the talented students who aim for national and international informatics Olympiads, GeoGebra offers programming in Java Script and GeoGebra Script. By using this framework, GeoGebra can effectively be applied in all subfields of Computer Science and

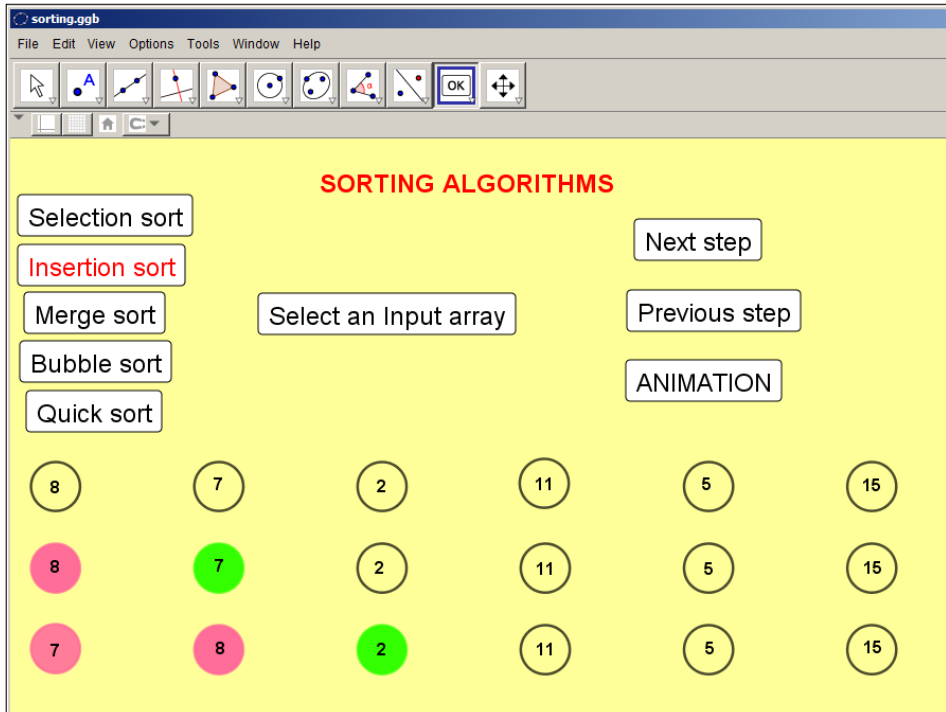


Fig. 2. GeoGebra animation for sorting algorithms.

Informatics to familiarize students with programming concepts, develop their computational thinking and facilitate their participation in competitions.

4.3. Examples of GeoGebra Applications

4.3.1. Horizontally Launched Projectile Simulation

Fig. 3 shows a GeoGebra simulation for a horizontally launched projectile, in this case, an apple. In this simulation, we practically demonstrate the multilingualism, an important feature that GeoGebra offers. Namely, the upper left corner of the Graphics view contains the buttons “MK”, “EN” and “RU” that symbolize the three languages, Macedonian, English and Russian, respectively. An active language indication is the red lettering label. The number of languages we can include in the simulation is unlimited.

Three sliders were used, for initial speed (v_0), initial height (h) and the time of flight (t) of the projectile. At whatever moment of the apple movement, the Graphics view displays the current coordinates of the apple. According to the values of the sliders shown in the Fig. 3, it is clear that the initial position of the apple has coordinates $(0, 7.5)$, and the initial speed of the apple is $v_0 = 10$. The Fig. 3 also displays the coordinates of the apple $(4.9, 6.3)$ after $t = 0.5$ s and its current horizontal distance $D = 4.9$ m. At whatever moment of the movement, the coordinates of the apple satisfy the formulas displayed on

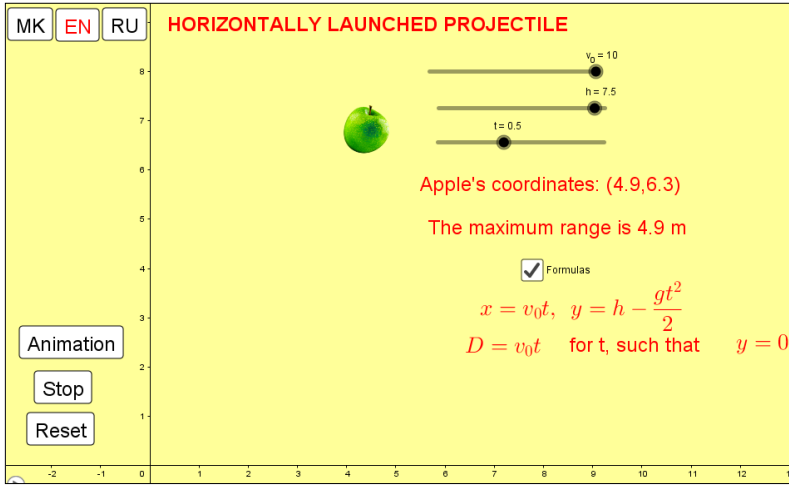


Fig. 3. GeoGebra simulation for horizontally launched projectile (English language version).

the lower-right part of the Graphics view. The “Animation”, “Stop” and “Reset” buttons control the apple’s motion in the projected path. Their meanings are “start movement”, “stop movement” and “set the apple at the starting position”, respectively.

4.3.2. Synchronization of Analog and Digital Settings Application

A little more sophisticated application is the “Synchronization of analog and digital settings” application (Fig. 4), which is intended for students in the lower grades in elementary schools.

Its elements are an analog clock on the left side, two check boxes (“Random Digital Time” and “Set Digital Time”) on the right side and two buttons (“Generate random analog time” and “Set the arrows”) below the analog clock.

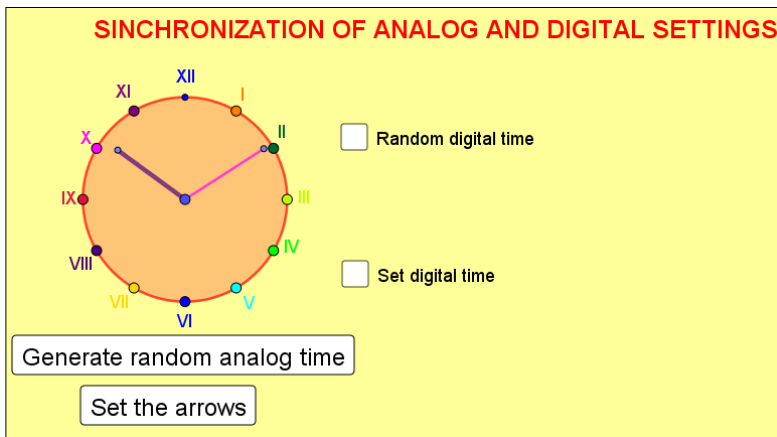


Fig. 4. Synchronization of analog and digital settings (version 1).

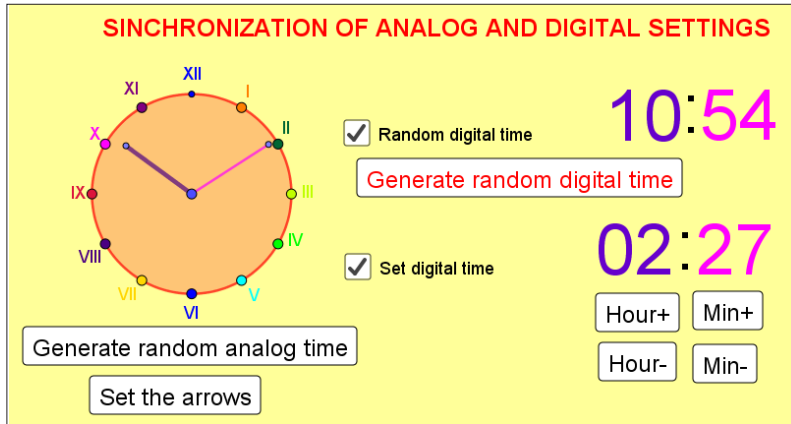


Fig. 5. Synchronization of analog and digital settings (version 2).

If we click on the first button, random time is automatically generated. More precisely, in the background of the application, random values from the interval $(0, 2\pi)$ for both the hour hand angle of rotation and the minute hand angle of rotation are selected. The center of rotation of the hour and the minute hands is the center of the clock (the point where the beginning of the two hands is fixed). Clicking the second button releases both the hour and the minute hand and they can be moved manually.

We can move the hands by clicking and dragging their ends, i.e. by rotating the hands about the center point of the clock. After clicking (“ticking”) on each of the “Random Digital Time” and “Set Digital Time” fields, two digital clocks are generated right next to them (Fig. 5).

Unlike the case with the analog clock, after clicking the “Generate random digital time” button, we generate four random numbers h_1 , h_2 , m_1 and m_2 , which represent the time $h_1h_2:m_1m_2$. The first two are the hour digits, and the second two are the minute digits. When the “Set Digital Time” checkbox is “checked”, it displays a digital time that the user can manually adjust by clicking on the the following buttons: “Hour+”, “Hour-”, “Min+” and “Min-”.

Thanks to the flexibility of GeoGebra, this application can serve as a basis for creating other applications. By combining the existing parameters (buttons, checkboxes), excluding some of them and/or including new parameters, we can create multiple applications, animations and simulations with different weights. The teachers who are familiar with these GeoGebra capabilities will be able to adjust the degree of application complexity and to enhance the individualization of teaching and learning.

There are lot of possible modifications of this application. We can set the clocks (analog and digital) to display the same time. Then, the speed of the animation can be adjusted to be equal to the speed of displaying the exact time of a real clock. In this way, the animation could serve as a classic clock. In another modification of this application, both clocks could serve as stopwatches. Possible upgrades to this modification include activating an alarm at a predefined time together with a sound file. To increase reality of

all possible versions of this application, we can activate an appropriate sound file in the background that will simulate the clock ticking sound effect.

Teachers can anticipate a full range of tasks related to the basic version of this application. Below is a list of possible general problem tasks that a teacher can set up:

- i) Generate a random analog time, then match the digital setting to the analog one.
- ii) Generate random digital time, then match the analog setting to the digital one.
- iii) Generate a random analogue time, then adjust the digital setting to be forward or backward for a certain number of hours and/or minutes in relation with the analog setting.
- iv) Generate a random digital time, then adjust the analog setting to be forward or backward for a certain number of hours and/or minutes in relation with the digital setting.

5. The GeoGebra Researches

In Fall 2018, we conducted three researches. The first research targeted primary and high school teachers regarding their knowledge, the frequency, the means and the purposes of their GeoGebra use during classes. The second research focused on first-year university students' answers to questions related to their GeoGebra use during their primary and high school years. The third research examined the effectiveness of GeoGebra's use through a comparison between two types of class groups, experimental (with use of GeoGebra) and control (without use of ICT) group. Through these researches we got a clear image of GeoGebra's use in schools, particularly the profile of GeoGebra's users and their dedication for using GeoGebra. We investigated the amount of knowledge users (students and teachers) have about GeoGebra through the method of self-evaluation and the extent to which GeoGebra animations, simulations and computer games can contribute to a better understanding of the learning material.

5.1. *The First Research*

In our first countrywide research, 207 teachers from 27 cities participated. The electronic questionnaire was created in Google Form and included 36 questions.

The allocation of teachers by subjects they teach is as follows: Mathematics (38%), Informatics (26.5%), Natural sciences (19%), Programming languages (17%), Work with computers and basics of programming (5.5%), Information technology (13%), Technical education (10.5%), Physics (9.5%), Biology (8%), Chemistry (7.5%), Electrical engineering (6.5%), Linear algebra and Analytic geometry (5%), Algebra (5%), Mathematical analysis (5%), and Geography (3%).

52.7% of total number of responders are secondary education teachers, 19.5% are lower primary school teachers, and 27.8% are upper primary education teachers.

Over 70% of the respondents have many years of experience in teaching (38.2% of them have more than 20 years of experience, 31.9% have 11 to 20 years of experience,

19.1% have 6 to 10 years of experience, 8.3% 2 to 5 years, and only 2.5% of the respondents have less than 2 years of experience).

The results of the teachers' self-evaluations indicated that 55.9% of them do not have any knowledge of this software. 20.4% of the respondents have a basic level of GeoGebra knowledge, 18.8% have medium level of knowledge, and only 4.8% of the respondents answered that they have advanced level of GeoGebra knowledge.

The results of the research also showed that GeoGebra is most often used for drawing function graphics (86.2%), for solving constructive tasks (48.3%), for displaying animations (48.3%), for displaying simulations (41.4%), for displaying geometric bodies and/or functions in three-dimensional space (25.9%) and for solving different types of equations (24.1%). From these responses and from the answers to the other questions, it can be concluded that teachers who use GeoGebra in schools, use only few opportunities offered by this software.

Approximately one quarter (24.5%) of the respondents do not use animations, simulations and/or computer games in the process of teaching, 32.8% of the respondents use them once or twice a month, 27% once to twice a week, and only 7.4% use them daily.

5.2. The Second Research

This research was conducted over a sample of 187 first grade university students who attend the bachelor program "Application of Information Technologies" at the Faculty of Information Sciences and Computer Engineering at the University "St. Cyril and Methodius" in Skopje. They had obtained a degree in secondary education in a total of 32 cities. The 19-questions questionnaire was filled in by the respondents on paper. The majority of the respondents, i.e. 67.9% completed high school education (Gymnasium), 20.9% of them secondary vocational education, and 9.1% of the respondents obtained a diploma from a secondary economic school.

About 72% of them stated that they did not use the GeoGebra software during their primary education, and over 56% did not use it during the secondary education. Out of all students who used GeoGebra software, 10.4% self-assessed themselves with the lowest grade "1", 23.5% with grade "2", 42.6% with grade "3", 20% with grade "4", and only 3.5% with grade "5".

5.3. The Third Research

The third research consisted of three parts. The first part consists of an experiment conducted in two Physics classes of first year high school students (one experimental and one control group). The second part consists of an experiment conducted in two Physics classes of third year students (one experimental and one control group) in the same high school as the first part. The third part consists of an experiment conducted in two Mathematics classes of fourth-year students (one experimental and one control group) in another high school. The experimental groups used GeoGebra simulations and the control groups had instruction conducted without the use of ICT tools.

Table 1
The main features of the third research

	First part	Second part	Third part
School title	Orce Nikolov	Orce Nikolov	Josif Josifovski Gevgelija
City	Skopje	Skopje	
Subject	Physics	Physics	Mathematics
High school year	First	Third	Fourth
Lesson title	Horizontally launched projectile	Polarization of light	Real functions
Total number of students	68	59	42
Number of students in experimental class	34	29	22
Number of students in control class	34	30	20
Simulation used (in experimental group)	Horizontally launched projectile	Partial and full polarization of the light (air-water case and air-glass case) and the Malus's law for light in-tensity	Graphics of: Square function, Sine function, Cosine function, Exponential function and a Logarithmic function

Afterwards, all experimental and control groups took a test (post-test) of the material from the previous lesson learned. In the end, we compared the results of students' previous test (pre-test) and the results achieved at the test (post-test) in both classes, in each of the three parts of the research. We took into account only the answers to the questions (tasks) closely related to the simulations used.

The results of the researches conducted using GeoGebra simulations, among other things, showed: a) identical average grades (4.05) from the pre-tests among students from both groups; b) a better average post-test grade in experimental groups of students (3.4 versus 3.26 in control groups), which was especially expressed in the results of the mathematics test; and c) the t-test was used during statistical processing of the data. We were considering Null Hypothesis $H_0: \mu_1 = \mu_2$ versus the Alternative Hypothesis $H_1: \mu_1 > \mu_2$. With a significance level of 0.0168, the average pre-test grade was better among the students from the experimental group, and with a level of significance 0.001 the same applies to the second test, whereby we conclude better success among the students who followed the instruction using the GeoGebra simulations. It is interesting to know that the statistical data processing was carried out by using the GeoGebra's tools.

6. General Conclusions and Recommendations

Taking into consideration all previous sections in this paper, if GeoGebra begins to be applied at all levels of education, there will be multiple benefits. First of all, it will modernize teaching, stimulate the creativity of students and their critical thinking skills, and will lead to easier adoption of 21st century skills among the students. Using GeoGebra in teaching will intensify the application of active learning methods,

such as Problem-based learning, Project-based learning, Learning with experimentation, Collaborative learning, etc. Secondly, students who start using GeoGebra at the beginning of their education will upgrade their knowledge more easily in the further levels of their education. Thirdly, if the GeoGebra software is installed in the schools, the problems of installation, use and upgrading that are inherent in many modern software tools will be minimized. This conclusion relies on the fact that GeoGebra does not have high technical requirements for the necessary hardware and software, it is easily installed on older computers, and even the most complex animations, simulations and computer games made in GeoGebra occupy minimum disk space, expressed in kilobytes. The use of GeoGebra will overcome the gap between opportunities provided by schools in rural and urban environments, as well as between well and poorly equipped schools. New possibilities for experimentation in teaching and learning will be opened, without the danger of having physical and health consequences, in comparison with the existence of classical laboratory experiments, which are available in a limited number of schools. In our opinion, GeoGebra is mostly beneficial to gifted and talented students, the unjustly neglected category of students in our education, who can play an important role in the development of our society. Usually, teaching is tailored to the level of an average student and gifted students do not receive the necessary attention and care from their teachers, thus the children's further education is left to the enthusiasm, motivation and knowledge of those teachers. With the help of GeoGebra, all students will be given the opportunity to explore the incredible field of experimentation. GeoGebra can also be used as a tool for preparing students for competitions in different fields.

The researches conducted suggest that there is a room for improvement regarding teachers' and students' knowledge of this software. The results of the experiments particularly show that these users do not make the maximum out of GeoGebra's features. A peripheral area that was investigated during the first research was exactly the teachers' willingness to improve their GeoGebra skills. 69.4% of all teachers who took a part of the first research expressed their interest in attending events and trainings which can further contribute to expanding their knowledge for GeoGebra. As a result of that, we would get more teachers knowledgeable in this field, thus students will be given a better opportunity to use GeoGebra animations and simulations frequently under their teachers' supervision. The fact that students from the third research who use GeoGebra achieve better results compared to students who do not have this habit shows that GeoGebra is a highly promising software whose potential can further be used by qualified teachers and motivated students.

The proposal for the selection of GeoGebra as a high-quality software solution that is recognized, accepted and used worldwide, at the same time satisfies two important criteria that are of great importance in societies (developing countries) like ours, namely, minimizing the costs and maximizing the quality. This proposal arises as a result of a part of the research and it relies on the commonly known benefits of the application of GeoGebra. GeoGebra is a free and open-source software tool and represent the ideal solution for our educational institutions, for students and their teachers.

References

- Bu L., Schoen R., (2011). *Model-Centered Learning: Pathways to Mathematical Understanding Using GeoGebra*, Sense Publishers, Rotterdam.
- Chrysanthou I. (2008). *The use of ICT in Primary Mathematics in Cyprus: The case of GeoGebra*, University of Cambridge, Faculty of Education, Cambridge.
- GeoGebra* (2019). <https://www.GeoGebra.org>
- GeoGebra Institutes* (2019). https://wiki.GeoGebra.org/en/Category:GeoGebra_Institute
- GeoGebra s* (2019). <https://help.GeoGebra.org/topics>
- GeoGebra Wiki Pool* (2019). https://archive.GeoGebra.org/en/wiki/index.php/Main_Page
- Hohenwarter, M., Lavicza, Z. (2010). *Gaining Momentum: GeoGebra Inspires Educators and Students*.
- Jancheski, M. (2019). Educational software, digital learning materials, and teaching and learning by using ICT under conditions of mass informatization in education, Skopje.
- Jancheski, M. (2019). Educational software for students from kindergartens and lower primary school. In: *13th International Technology, Education and Development Conference, Valencia, Spain. Proceedings*. Published in INTED2019. 7603–7612
- Jancheski, M. (2019). GeoGebra animations, simulations, and computer games in teaching and learning science. In: *13th International Technology, Education and Development Conference, Valencia, Spain. Proceedings*. Published in INTED2019. 7613–7623
- Klllogjери, P., Klllogjери, Q. (2014). *GeoGebra: A vital bridge linking mathematics with other sciences (available to everyone)*.
- Leggett, A. (2014). *Active learning pedagogies: Problem-based learning*.
- Mukiri, M.I. (2016). *Feasibility of Using GeoGebra in Teaching and Learning*. Department of Educational Communication and Technology, Kenyatta University.
- Stephen Jull's LinkedIn page (2018). <https://ca.linkedin.com/in/stephenjull>



M. Janceski, Ph.D., has more than two decades teaching and research experience at the Faculty of Computer Science and Engineering, under the “Ss. Cyril and Methodius” University in Skopje. Leader of the Macedonian team on 16 BOI and IOI competitions. He has participated in 19 scientific projects, including the crucial ICT projects in Macedonian education, as consultant and trainer. Author of numerous scientific papers, manuals, and textbooks. His major research areas are Distance Education, ICT in Education, Educational Software and Didactics.



S. Jancheska is a Computer Science student in New York University Abu Dhabi. She has successfully participated in “Infomatrix International Computer Project Competition” and “Golden Climate International Environmental Project Olympiad” winning one bronze and two silver medals. Passionate about educational softwares that facilitate students’ learning among which is GeoGebra.

Survey and Analysis of Computing Education at Japanese Universities: Non-IT Departments and Courses*

Tetsuro KAKESHITA, Mika OHTSUKI

*Faculty of Science and Engineering, Saga University, 840-8502, Saga, Japan
e-mail: kake@is.saga-u.ac.jp, mika@is.saga-u.ac.jp*

Abstract. We conducted the first national survey of computing education at Japanese universities in 2016. In this paper, we report the survey result of the computing education at non-IT departments and faculties whose major subject is not computing. The survey covers various aspects of computing education including program organization, quality and quantity of educational achievement, students, teaching staff and computing environment. We collected 994 answers through the survey. At least 87,000 non-ICT students are taking computing education in Japan. Although computing education is carried out at every major academic discipline, teaching effort greatly differs depending on the academic discipline. We also find shortage of teaching staff for computing education. The analysis result will be an essential input to develop reasonable curriculum guidelines and accreditation criteria to improve computing education at non-IT departments.

Keywords: computing education, web-based survey and analysis, college level education, quality assurance in education.

1. Introduction

Information Technology (IT or ICT) is regarded as an essential infrastructure of the modern society. IT is also expected as a driver for business and/or social innovation at many countries. For example, EC refers to such skill as e-Skills and works on promoting the development of e-Skills in EU countries (EC, 2007). College level computing education is essential to develop citizens and IT professionals having enough knowledge and skill on IT. Such computing education is required for students whose major is not

* This paper is a revised and extended version of the following paper written by the same author. T. Kakeshita, "National survey of Japanese universities on computing education: Analysis of non-IT departments and courses", in Proc. 12-th International Conference on Digital Information Management (ICDIM 2017), 81-86, 2017.

IT (Urban-Lurain, 2000) as well as for the students majored in IT. Furthermore, many countries including Japan are recently starting computing education from elementary school (Computing at School, 2008; K-12 Computer Science Framework).

Considering the above background, computing education is essential at modern universities. There are four types of computing education in Japanese universities. The situation is expected to be the same at other countries.

- A. Computing education at a department or a course majored in computing discipline.
- B. Computing education at a non-IT department or a course, whose major is not IT or computing, as a part of their major field of study.
- C. General computing education for all students at a university or a faculty typically at the first or second academic year.
- D. Computing education for the students willing to obtain high school teacher license on computing subjects.

We conducted a national survey of Japanese universities on computing education in 2016 (Kakeshita, 2017). The survey is composed of five survey types A through E. Among them, survey types A to D correspond to each type of computing education described above. The survey type E is executed for educational computer system which is a fundamental infrastructure for various types of computing education. Our survey is actually the first national survey on this subject in Japan, since there was no widely accepted definition of computing education.

The Science Council of Japan announced the reference standard of informatics (Hagiya, 2015) for university education in March 2016. The reference standard provides a common body of knowledge (BOK) for college level computing education so that the Japanese government accepted this as a definition of computing education.

In this paper, we report and discuss the result of the survey type B for computing education at non-IT departments and courses. The purpose of this survey is to analyze and understand the current achievement of computing education at Japanese universities from various aspects including program organization, quality and quantity of educational achievement, students, teaching staff and computing environment.

The ultimate goal of this research is to develop reasonable curriculum guidelines and accreditation criteria to improve computing education at non-IT departments. Fundamental understanding of the achievement of computing education is necessary to achieve this goal. Such effort is necessary since the importance of computing education is increasing in the modern society.

We have already published the survey outline in (Kakeshita, 2017). The results of other survey types were published separately (Kakeshita, 2018; Ohtsuki, 2017; Sumi, 2017; Takahashi, 2017). Information processing society of Japan (IPJS) utilizes the survey result to develop the new J17 curriculum standard (Information Processing Society of Japan, 2018) for computing education in FY2017. The Japanese Ministry of Education (MEXT) will utilize the survey result to improve the national policy of computing education.

2. Related Work

International or nationwide comprehensive surveys on the status of some educational subject tend to be carried out regarding rather well-established subjects such as mathematics and science than relatively new subject as computing and informatics.

TIMSS (Trends in International Mathematics and Science Study) (TIMSS & PIRLS) was firstly executed in 1995, and is one of the representative international surveys aiming at evaluating educational outcomes on mathematics and science domain at elementary and secondary levels. The TIMSS survey contains inquiry into the status of pupils and students' achievement and national curriculums of mathematics and education as well. ACT National Curriculum Survey (National Curriculum Survey) is an example of the nationwide surveys which investigate curriculums of several subjects, such as English language, arts, mathematics, science, that also appear to be well-established as educational subjects.

On the other hand, some examples of the surveys related to computing education are found, however their focus were mostly specialized on some limited aspects of education rather than entire picture of curriculum execution as we presented in this paper.

For example, (Goldweber *et al.*, 2011) reported how social issues of computing were included into computing curricula referring to an international survey of computing instructors. Simon *et al.* (Simon, *et al.*, 2018) presented an examination of the choice of the programming language in introductory programming courses based on parallel surveys conducted at Australian and UK universities. Marshall (Marshall, 2012) showed a comparison of the core aspects of the ACM/IEEE Computer Science Curriculum 2013 with the specified core of CC2001 and CS2008 to identify the changes of the curriculum. This kind of curriculum survey is in common with our survey in terms of their holistic viewpoints. However, the survey we conducted was about the 'actual execution' of the curricula in several universities placed at different countries, which gave unique nature to the survey we conducted.

Through the literature review, we came to find that our survey and comparative analysis have some specific features compared with the related works, and add original value to the survey.

The most apparent features is the comprehensiveness. For instance, the questionnaire of the survey, as we see in the next section, contains both the questions about educational achievement and those about program overview as well.

We have found another example of international survey on educational content concerning computing and informatics domain (Al-Ansari, 2002). However, its focus was entirely on the educational achievement aspect in our term. The survey which was done focusing on both the aspect of computing curriculum (which was covered by educational achievement) and that of educational environment and human engagement (which was covered by program organization etc.) in one time is very unique among relevant surveys.

3. Survey Outline

3.1. Survey Questions

The following is the list of questions for survey type B. The list shows that our survey covers various aspects of computing education by considering the Japanese standard for establishment of universities and our experience of accrediting computing programs in Japan:

- Name of university, faculty, department and course.
- Program organization:
 - Day time, night or remote program.
 - Academic discipline of the program such as engineering, social science and humanities.
 - Required number of credits of computing subjects for graduation.
 - Number of computing subjects provided.
- Quality and quantity of educational achievement:
 - See Section 3.2 for detail.
- Enrolled students:
 - Regular academic years of computing education.
 - Number of students.
 - Student's choice of career after graduation.
- Teaching staff:
 - Number, educational background, current specialized field, tenure of faculty members.
 - Number and workload of support staff and teaching assistant students.
- Computing environment:
 - Educational computer system.
 - Student's own personal computer (PC) and its utilization at class.
 - Educational programming language.
- Other topics:
 - Future plan and strength of the program.
 - Utilization of IT certification and/or qualification.
 - Special remarks.

3.2. Survey of Quality and Quantity of Educational Achievement

The survey of quality and quantity of educational achievement is the core of our survey. We define six achievement levels for knowledge and skill represented in Table 1. These levels are used to describe quality of education.

We also define a BOK based on the reference standard of informatics (Hagiya, 2015) and additional topics related to general computing education (Kawamura, 2008). The BOK contains 90 topics classified by 21 domains as represented in Table 2. The BOK is

Table 1
Knowledge and Skill Level Definition

Level	Knowledge Level Definition	Skill Level Definition
0	Not taught (unnecessary or already taught at general computing education)	
1	Not taught because of the time limitation or because the level of the contents is too high	Taught at class with simple exercise
2	Taught at class. Students know each term	Taught at class with some exercise. Students can perform the topic if detailed instruction is provided.
3	Taught at class. Students can explain the meaning of each term	Taught at experiment with more complex exercise. Students can perform the topic with simplified instruction
4	Taught at class. Students can explain relationship and/or difference among related terms	Students perform combined research project containing the topic so that the students can autonomously perform the topic
5	Taught at class or graduation research project. Students can teach related domain or subject of the terms to the others	Students perform combined research theme containing the topic. Students can teach how to perform the topic to others

Table 2
Common BOK Organization

Source	Section	Domain
J07-GEBOK	General Education	Informatics in General Education (9)
Reference Standard of Informatics	(A) General Principles of Information	(6)
	(B) Principles of Information Processing by Computers	Information Transformation and Transmission (4), Information Representation, Accumulation and Management (4), Information Recognition and Analysis (4), Computation (6), Algorithms (8)
	(C) Technologies for Constructing Computers that Process Information	Computer Hardware (3), I/O Device (4), Fundamental Software (3)
	(D) Understanding Humans and Societies that Process Information	Process and Mechanism for Information Creation and Transmission (2), Human Characteristics and Social System (3), Economic System and Information (2), IT-based Culture (2), Transition from Modern Society to Post Modern Society (2)
	(E) Technologies and Organizations for Constructing and Operating "Systems" that Process Information in Societies	Technics for Information System Development (7), Technics to Obtain Information System Effect (6), Social System Related to Information (2), Principle and Design Methodology for HCI (4)
	Competence	Professional Competency for IT Students (3), Generic Skill for IT Students (6)

used to precisely define educational achievement of each program. The numbers within the parenthesis are the number of topics belonging to the section or the domain.

We adopted the same definition of level and BOK throughout the survey types A to D in order to enable mutual comparison of the different survey types. Such comparison is important to understand relationship among different survey types.

3.3. Survey Process

We prepared the survey in October 2016. We defined the survey questions and set up the web-based survey system (Kakeshita, 2011). We utilized the web-based survey system since we did not exactly know the actual organization for this survey in advance. After preparing various documents such as user manual and detailed explanation of the survey questions, we sent the formal request letter to all universities in Japan with a reference letter from the Japanese Ministry of Education in order to increase the response rate.

The survey was executed for two months starting at the beginning of November 2016. Each survey responder must first register to the web-based survey system and then answer the questions listed in Sections 3.1 and 3.2. We also provide FAQ and independent answers for the questions from the responders.

Each user answers to the survey questions listed in Section 3.1 through a web-based answer sheet as illustrated in Fig. 1. Although the answer sheet is prepared for the survey type A, the answer sheet for the survey type B is similar except that the questions are slightly different. The questions for each survey type can be easily customized by setting up the master database.

Each user answers to the survey of quality and quantity of educational achievement defined in Section 3.2 by uploading an Excel worksheet as illustrated in Fig. 2. Each survey responder is requested to fill the blue cells where each row respectively represents knowledge and skill achievement levels, and the number of enrolled students learning the specified topic.

After closing the survey, we reviewed the collected answers and requested the responders for possible correction of the incomplete or inconsistent answers.

A : 情報専門学科新規登録

ユーザーIDから対象組織までの値を入力して画面下部の「新規登録」ボタンを押してください。項目1.1以降の値は後でも変更できます。

● は必須項目です。

User ID	<input type="text"/>	メールアドレスを入力してください。(携帯メールアドレス不可)
Password	<input type="password"/>	パスワードを入力してください。「!」「*」「 」「,」「<」「>」は入力禁止文字です。
ユーザー名	<input type="text"/>	確認の為、再度パスワードを入力してください。
	<input type="text"/>	氏名を入力してください。
対象組織	「調査項目に関する詳細説明」を参照の上、対象組織を指定してください。リストに無い場合は「登録」を選択し入力してください。 大学名 <input type="text"/> ● 登録 学部名 <input type="text"/> ● 登録 学科/課程名 <input type="text"/> ● 登録 コース名等 <input type="text"/> ● 登録	
Questions	<input type="text"/> ● 登録	
1.1 昼夜別	<input type="text"/> 昼間, 夜間, 通信制の区分 (複数の区分の教育プログラムを提供している場合は、個別に回答)	
1.2 対象領域	<input type="text"/> 工学 学校基本調査の区分に基づく対象領域	
1.3 107専門領域	<input type="text"/> CS (コンピュータ科学) 情報処理学会107カリキュラム標準の区分に基づく専門領域	

Fig. 1. Web-based Answer Sheet for Each User.

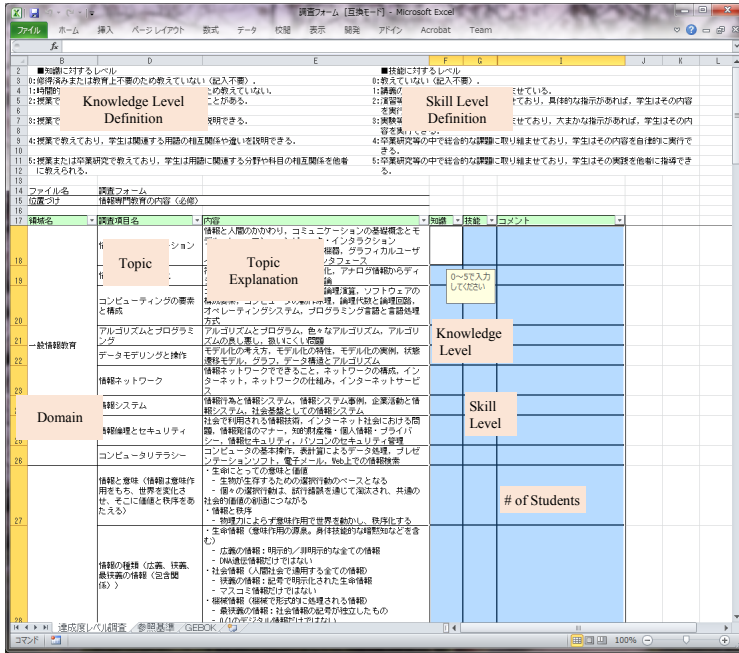


Fig. 2. Answer Sheet using Excel Worksheet.

4. Overview of the Survey Result

The target of survey type B is a department or a course whose major subject is not in the computing discipline. Computing education at such department or course is composed of the general computing education, usually provided by a faculty or a university, and specialized computing education provided by the target department or the course as their major subjects. Survey type B is focus on the specialized computing education. General computing education is analyzed through survey type C (Takahashi, 2017).

4.1. Number of Responses

Table 3 represents the number of courses, departments and faculties (including universities) responded to the survey type B. The public universities are the universities run by a local government such as a prefecture or a city.

As can be found from Table 3, we allow a faculty or a university to respond to the survey type B. This is because that the faculty or the university can merge responses from the courses or the departments, since many non-IT departments or courses are expected to provide specialized computing education in the university or faculty.

Table 3.
Number of Responses to Survey Type B

Univ. Type	Course	Department	Faculty or Univ.	Total
National	62	173	67	302
Public	12	34	18	64
Private	67	452	109	628
Total	141	659	194	994

4.2. Student Enrollment Classified by Major Field of Study

The school basic survey utilizes 11 major academic disciplines to classify college level education (MEXT, 2017). Table 4 represents the number of students collected through the survey.

87,261 students (13.9% of the university students) are taking specialized computing education. We also find that 61% of the students are taking specialized computing education at the responded departments etc. Although there are many departments not responding to the survey, we can estimate that at least 100,000 students are taking specialized computing education as a part of their major field of study in Japan. Table 4 shows that all major disciplines provide specialized computing education. This fact indicates the importance of computing education.

Table 4
Number of Students Classified by Major Field of Study

Academic Discipline	# of Students		A/B (%)
	A*	B**	
Humanities	4,568	88,246	5.2
Social Science	31,428	204,933	15.3
Physical Science	4,969	18,523	26.8
Engineering	23,151	88,062	26.3
Agriculture	1,824	18,042	10.1
Health (Medicine and Dentistry)	3,438	11,765	29.2
Health (Others)	5,734	58,824	9.7
Home Economics	926	46,475	5.6
Education	2,599	17,787	5.2
Arts	645	18,189	3.5
Others	7,979	56,019	14.2
Total	87,261	626,865	13.9

* A: Number of students taking specialized computing education at the responded departments etc.

** B: Number of students collected through FY2016 school basic survey (MEXT, 2017).

Another observation from the table is that the ratio of the number of students taking specialized computing education divided by the total number of students greatly differ depending on the major field of study. The ratio indicates the degree of importance of computing education at each discipline. The importance is higher at the departments majored in engineering, physical science and health (medicine and dentistry). We consider that general computing education plays the major role in computing education at the academic disciplines with a lower ratio.

4.3. Number of Credits for Computing Subjects

7,883 computing subjects are provided by the responded departments. Among them 5,385 (68.3%) are lectures and 2,498 (31.7%) are exercises. This suggests a realistic ratio of the lecture and exercise to design a computing curriculum recommendation for non-IT departments. 390 departments (33.9% of the responded departments) provide 1 to 4 computing lectures and exercises. While 682 departments provide computing lecture, 316 departments (31.7%) do not. For the case of exercise, 426 departments (42.7%) do not provide any exercise.

Fig. 3 represents the distribution of required number of credits for the computing subjects for each academic discipline. The distribution is illustrated using box plot. The left

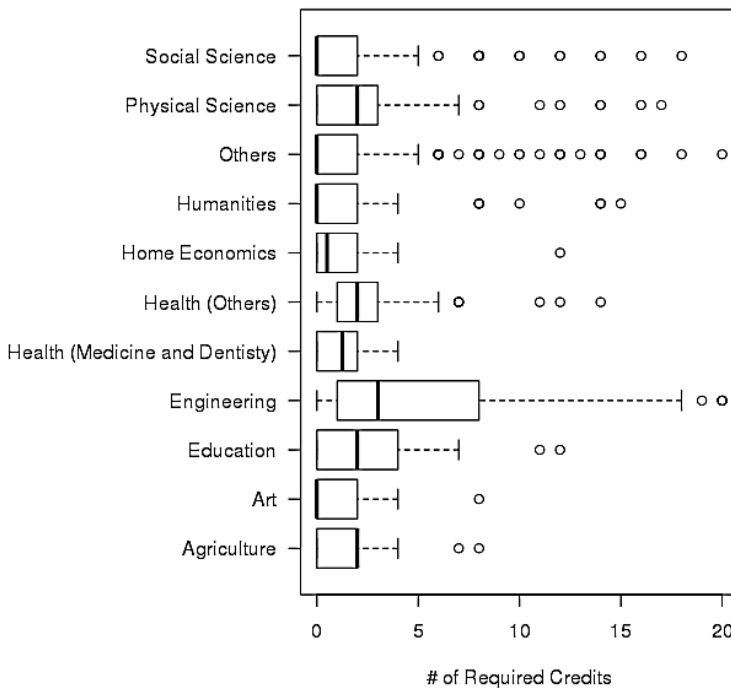


Fig. 3. Number of Required Credits for Computing Subjects.

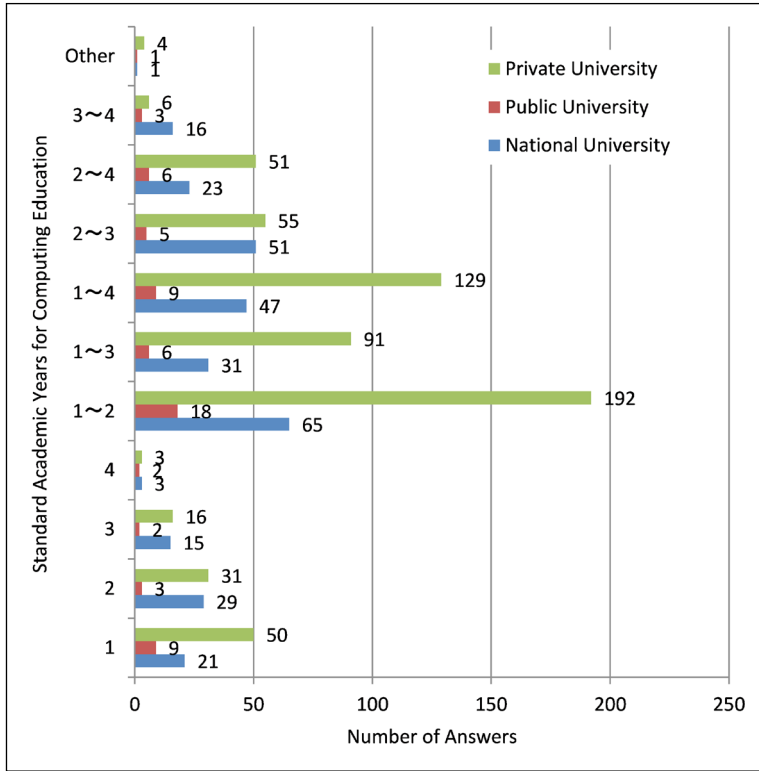


Fig. 4. Standard Academic Years for Computing Education.

and right sides of a box represent lower and upper quartiles of the collected data. The thick line represents the median. The left and right ends of the dashed line respectively represent lower and upper whiskers. Note that some of these values may coincide in the figure. The circles represent outliers. The distribution provides a realistic restriction to design computing curriculum for each academic discipline. For example, typical computing curriculum at non-IT departments is composed of 0 to 5 required credits. It is recommended to design a computing curriculum between 2 to 6 credits depending on the academic discipline to design a widely-accepted one.

Fig. 4 represents distribution of standard academic year for computing education. Computing education at non-computing departments typically starts at the first or second academic year and continues for two to four years. This tendency is essentially the same among national, public and private universities.

5. Educational Achievement

We shall analyze the educational achievement, i.e. quality and quantity of education, in this section. We collected 141 answers of the educational achievement. After classify-

ing the answers for each major field of study, we find that the number of answers is less than or equal to 2 in the case of domestic science, health (medicine and dentistry) and art. Thus we decided to analyze the educational achievements for the major fields other than these three fields.

We define effort of an educational program for a certain topic of the BOK by the multiplication of average level value and the number of students learning the topic. We thus define two types of effort values to teach knowledge and skill.

Fig. 5 represents knowledge effort classified by major field of study. Similar distribution can be obtained for the skill effort. The distribution represents focus of computing education at each academic discipline so that it is recommended to design a curriculum guideline considering the distribution of effort for each BOK section defined in Table 2. The figure is also useful to analyze difference of educational needs for computing education among the disciplines.

Fig. 6 illustrates the cluster dendrogram of the academic disciplines. The dendrogram is computed using hierarchical clustering using similarity of the disciplines. The difference of the heights between the disciplines represents the similarity of the disciplines. The similarity is calculated using the Euclidean distance of the effort distribution of the disciplines. Distance between two clusters is estimated using the complete linkage, i.e. maximum distance of all element pairs of the both clusters. For example, engineering and physical science are most similar so that we can develop a common computing curriculum for these two academic disciplines.

We shall next analyze educational achievement at each discipline. Fig. 7 represents the distribution of the total number of enrolled students for each BOK section and academic discipline. The numbers of the enrolled students are calculated by the sum of the number of enrolled students at each topic of the corresponding BOK section and academic discipline so that the actual values contain double counting of the same student. However we can observe that the disciplines of engineering, social science, and others

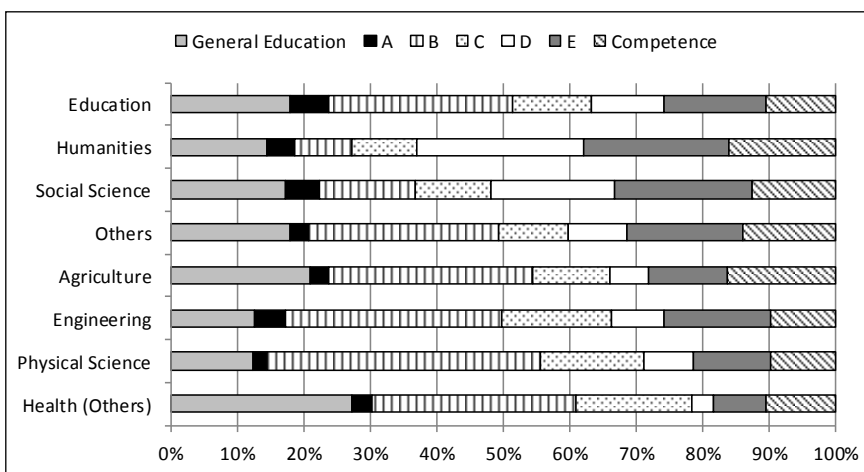


Fig. 5. Knowledge Effort Classified by BOK Section and Academic Discipline.

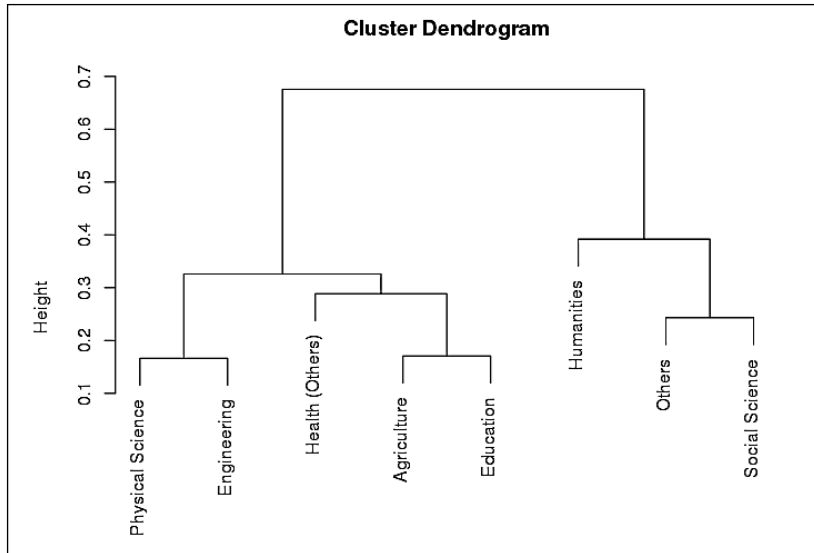


Fig. 6. Cluster Dendrogram of the Academic Disciplines.

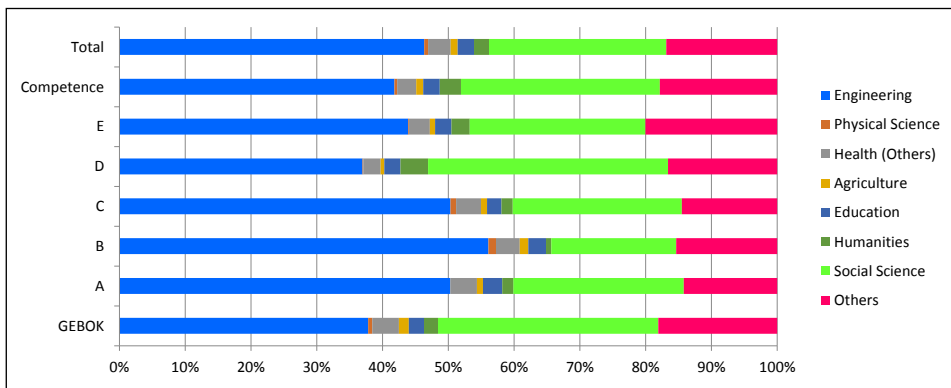


Fig. 7. Comparison of Academic Disciplines on the Total Number of Enrolled Students for Each BOK Section.

are the three largest disciplines of computing education and teach approximately 90% of the students. We shall call these disciplines as major disciplines in this paper.

Fig. 8–Fig. 15 represent average achievement levels (knowledge and skill) of each academic discipline for each BOK section. These figures are useful for each discipline to determine realistic levels for computing education at each BOK section. The readers can refer to Table 1 for the definition of levels.

We can observe that the achievement levels of the three major disciplines are not high compared with the achievement levels of the non-major disciplines. This is because that major disciplines contain various education programs and some of them cannot achieve

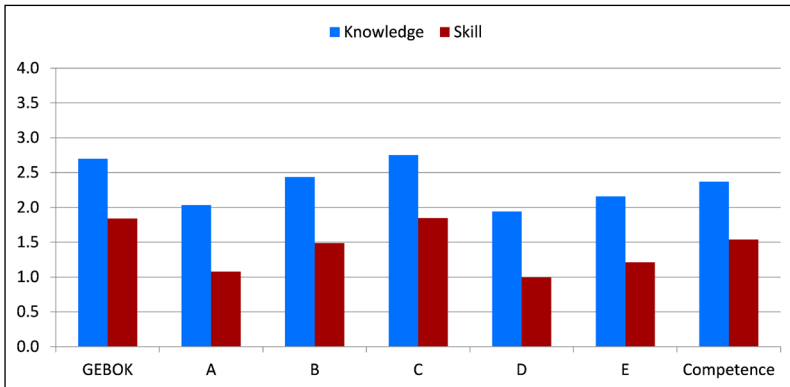


Fig. 8. Average Achievement Levels: Engineering.



Fig. 9. Average Achievement Levels: Physical Science.

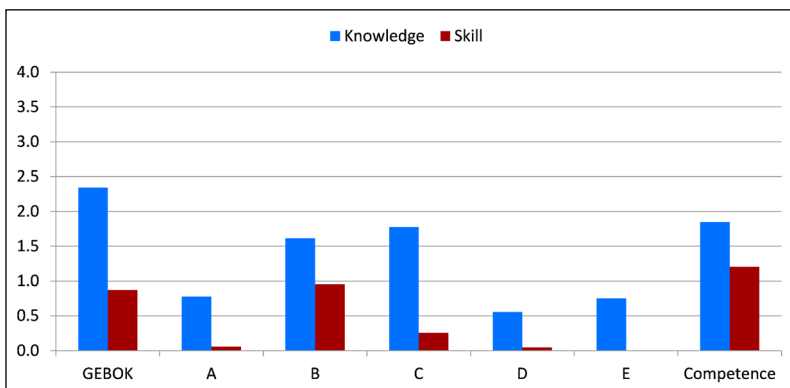


Fig. 10. Average Achievement Levels: Health (Others).

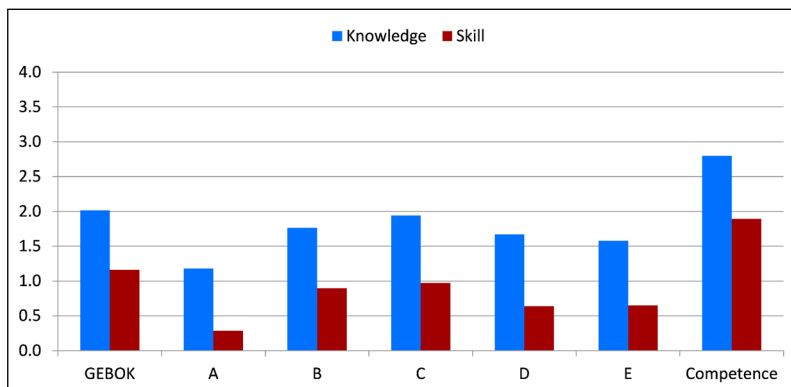


Fig. 11. Average Achievement Levels: Agriculture.

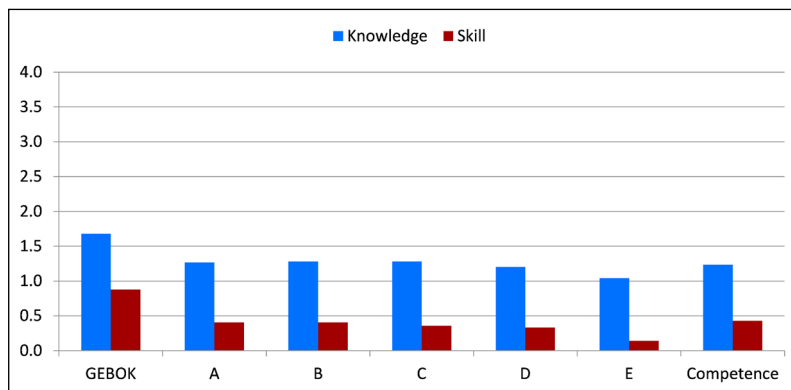


Fig. 12. Average Achievement Levels: Education.

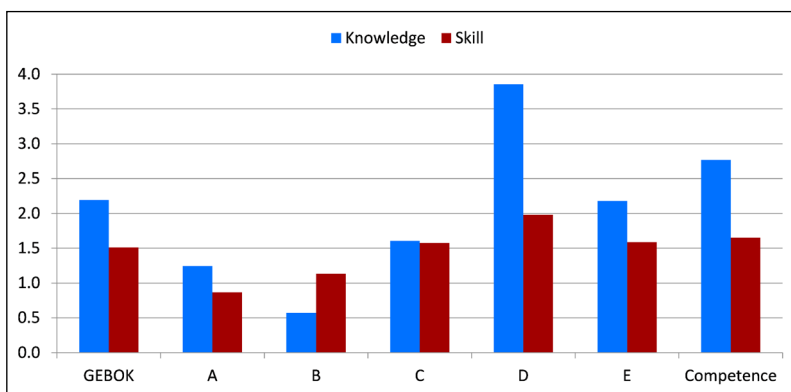


Fig. 13. Average Achievement Levels: Humanities.

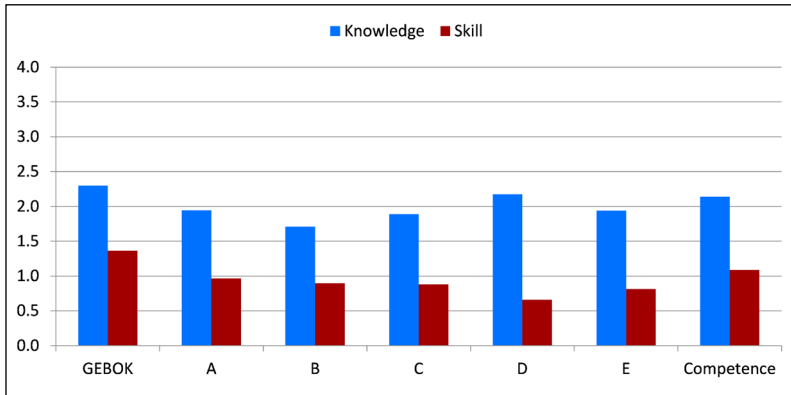


Fig. 14. Average Achievement Levels: Social Science.

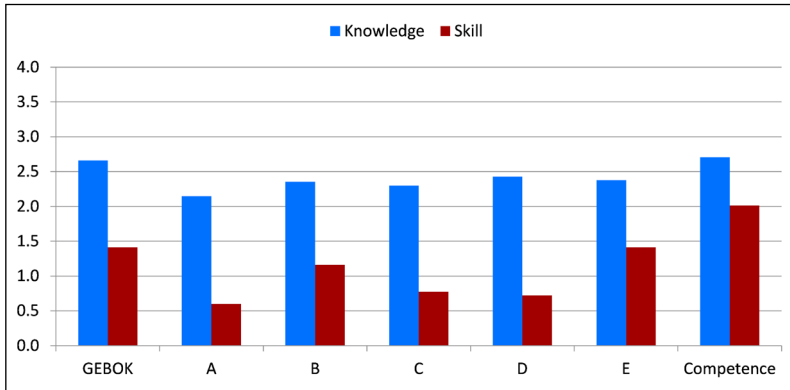


Fig. 15. Average Achievement Levels: Others.

high levels due to restriction of teaching staff and/or budget. On the other hand, some of the computing education at non-major disciplines achieve higher levels at a specific BOK section because they can concentrate education resources for the BOK sections.

The readers can also observe some similarity of the achievement level distribution between the similar disciplines illustrated in Fig. 6.

6. Enrolled Student

6.1. Distribution of Student Enrollment

Fig. 16 represents the distribution of student enrollment for the specialized computing education. The number of enrolled students indicates the upper bound of the

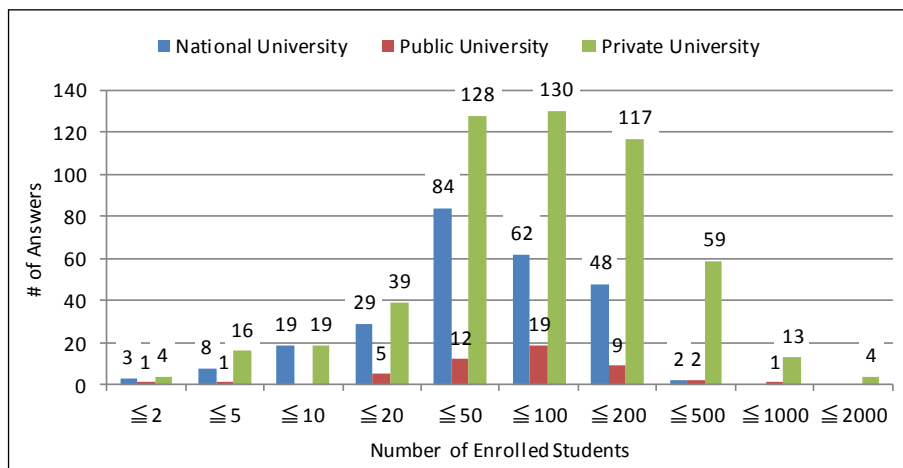


Fig. 16. Distribution of Student Enrollment.

numbers. For example, “ ≤ 20 ” means that the number is more than 10 and not more than 20.

The average number of enrolled students is 70.0 for national university, 87.3 for public university and 123.0 for private university. It can be observed that the number of enrolled students is larger at private university. In fact, 36.5 % of the private university has more than 100 enrollments.

6.2. Number of Students per Teacher

Fig. 17 represents the distribution of the number of students per teacher for the computing subject. The distribution greatly changes depending on the academic disciplines. The distribution is valuable to define accreditation criteria for the number of teachers for the computing subject. It will be reasonable to define the criteria at the lower 25% value of the distribution. If an educational program achieved better than the higher 25%, then it will be evaluated as a strong point of the program.

6.3. Student's Choice of Career after Graduation

Table 5 represents the student choice of career after graduation.

Since very small number of students go to graduate school majored in computing discipline, college level computing education ends at the undergraduate level. Although 13.8 % of the students go to a graduate school, the percentages greatly change at national and private universities.

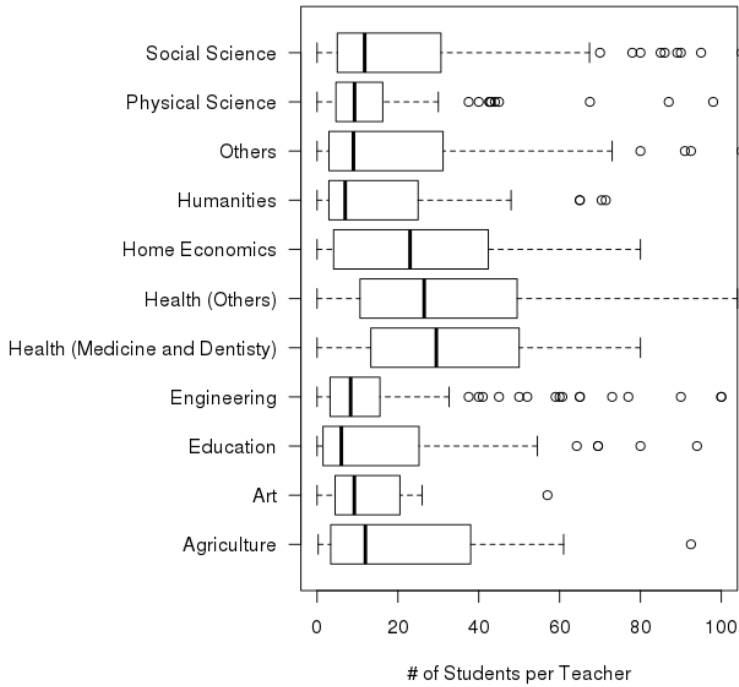


Fig. 17. Distribution of the Number of Students per Teacher.

Table 5
Student's Choice of Career after Graduation

University Type	Enter Graduate School		Get Job	Unknown
	Computing	Others		
National	443	9,270	15,969	2,498
Public	185	1,107	4,388	364
Private	147	4,757	66,911	9,603
Total	775	15,134	87,268	12,464

7. Teaching Staff

7.1. Faculty Member

Fig. 18 represents the number of faculty members teaching computing subject classified by the type of the faculty member and the university. The numbers shown in the bars represent the actual number of faculty members.

8,851 members are employed for specialized computing education. Full-time member ratio is higher at national and public universities. In fact, the ratio of part-time mem-

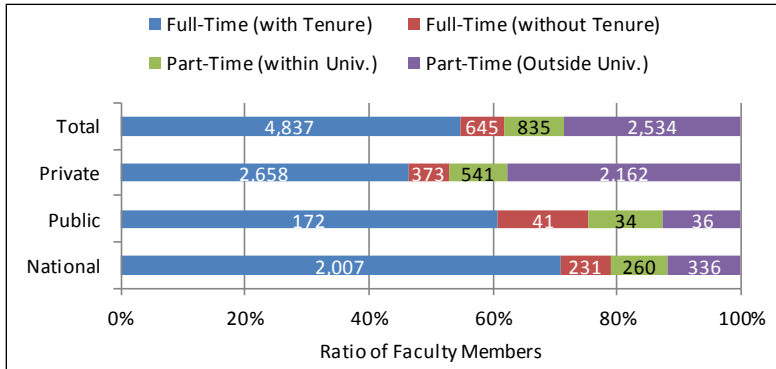


Fig. 18. Ratio of Faculty Members Teaching Computing Subject.

bers outside of the university is 37.7 % at private university. This is mainly because of the financial restriction and the restriction of full-time member post.

15,865 computing classes are held at each year. Full-time faculty members are in charge of more than 80% of the computing classes at national university as represented in Fig. 19. However, the ratio the part-time teachers outside of the university exceeds 25% at public and private university.

It is essential for the faculty members to have enough ability in the computing discipline to effectively teach students. We collected the number of computing department graduates and the number of faculty members whose current major is in the computing discipline. Fig. 20 represents the ratio of these two types of faculty members.

The ratio of computing department graduates is generally low in the four cases. The following is a list of the major reasons:

- (1) The number of Ph.D. holders in computing discipline is far less than the required number of faculty members to teach computing subjects.
- (2) Research contribution to the major field of the department is more important to hire a new full-time member than teaching ability of computing subject.

On the other hand, the ratio of faculty members majored in computing discipline is generally higher than the ratio of computing department graduates. This can be considered as an effect that the faculty member changed his/her major after getting position at the department and being assigned some computing subject.

7.2. Support Staff and Teaching Assistant

Table 6 represents the statistics of the support staff and teaching assistant (students to assist computing subjects).

It can be observed that teaching assistant is essential at many universities since the number of support staff is quite limited. Although most of the teaching assistants are the students of the employing university, students of the neighboring universities are also employed at a metropolitan area.

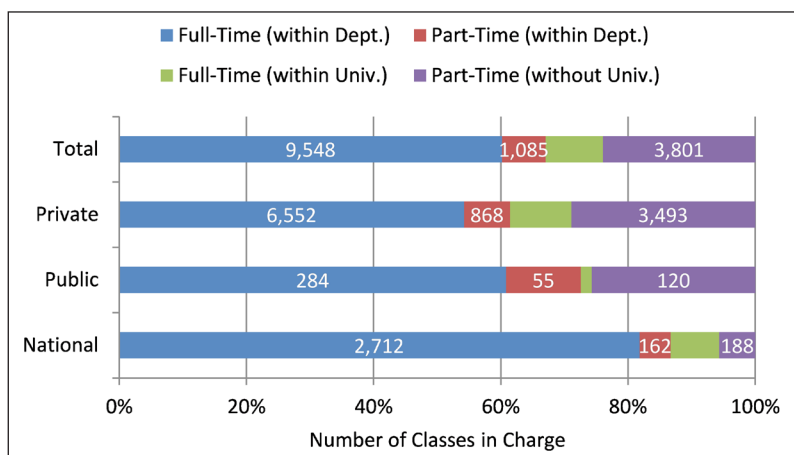


Fig. 19. Distribution of the Number of Computing Classes in Charge.

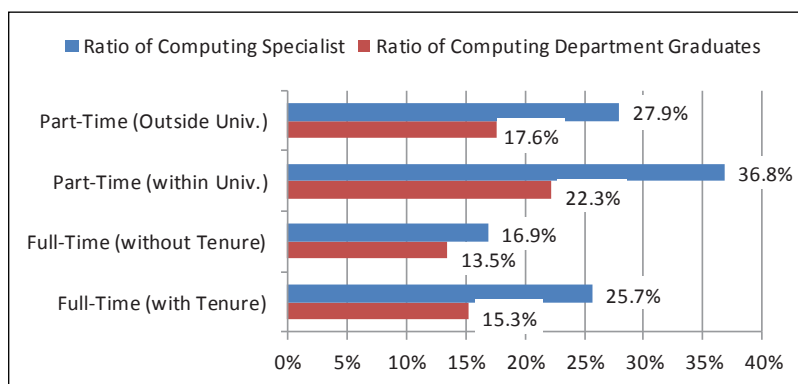


Fig. 20. Ratio of Computing Department Graduates and Faculty Members Majored in Computing Discipline.

Table 6
Support Staff and Teaching Assistant for Computing Subject

Univ. Type	Support Staff		Teaching Assistant	
	# of Staffs	# of Subjects	Workload (man hour)	# of subjects
National	166	74	42,390	818
Public	3	4	13,785	111
Private	434	432	73,125	1,889
Total	603	510	129,300	2,818

8. Computing Environment

8.1. Educational Computer System and Student PC

Educational computer system is important for effective computing education. Utilization of student PC for computing education is also important as the PC is getting cheaper. Table 7 and Table 8 respectively represent utilizations of educational computer system provided by the educational institution and the utilization of student PC.

We observe that 23.1% of the national universities do not have educational computer system in the university. This ratio is even larger in the cases of public universities (34.3 %) and private universities (37.6 %). 57.6 % of the universities utilize shared computer system.

80.6 % of the universities leave the decision to purchase PC to their students. Although PC price is getting cheaper, it is still difficult for many universities to impose obligation to the students to purchase PC.

We also find that 28.4 % of the departments have no educational computer system and allow students to decide to purchase PC. We need further investigation to these departments. On the other hand, 7.0 % of the departments have educational computer system and require students to purchase PC. We expect that these departments provide effective computing education by utilizing the educational computer system and student PC.

Table 7
Utilization of Educational Computer System

Utilization	# of Answers	# of Enrolled Students
Shared Computer System at University	356	38,148
Shared Computer System at Campus	141	12,839
Shared Computer System at Faculty	69	6,298
Private Computer System at Department	59	4,304
Computer System is provided but unused	43	4,201
No Educational Computer System	326	21,471
Total	994	87,261

Table 8
Utilization of Student PC

Utilization	# of Answers	# of Enrolled Students
All Students of the University must have PC	69	4,384
All Students of the Faculty must have PC	34	3,494
All Students of the Department/Course must have PC	26	2,335
Students are recommended to purchase PC	65	4,744
Purchasing of Student's own PC is optional	800	72,304
Total	994	87,261

Table 9
Popular Educational Programming Languages

Programming Language	National University	Public University	Private University	Total Score
C	174	38	254	466
Visual Basic/VBA	57	11	186	254
Java	40	4	102	146
C++	41	4	63	108
JavaScript	9	2	66	77
Fortran	34	2	27	63
SQL	8		23	31
Python	8		21	29
Ruby	6		19	25
PHP	6		16	22
R	13	4	3	20
Processing	3	3	9	15
Assembly Language	7		6	13
Matlab	7		4	11

8.2. Educational Programming Language

We collected three educational programming languages from each department with the highest achievement levels. Table 9 illustrates popular programming languages for the specialized computing education calculated using the collected data. The score of each language is evaluated as a weighted sum of the answers. The weight of a language is defined using the achievement level of the students at each department.

9. Concluding Remarks

We find that more than 100,000 students are learning computing subjects at non-IT departments or courses. The actual number of students would be even larger. Specialized computing education is carried out at all academic disciplines, which indicates importance of the computing education. We also find that the effort for the computing education is greatly different depending on the academic disciplines. The findings explained in Sections 4 and 5 will be useful to develop realistic curriculum guidelines for computing education at non-IT department or course. We also find shortage of teaching staffs specialized in the computing discipline.

Information Processing Society of Japan (IPSJ) published J17 curriculum standard for computing education in March 2018 (Information Processing Society of Japan, 2018). Since we find the importance of computing education at non-IT departments and courses through the survey, we intend to start a project to discuss about effective and feasible computing curriculum for non-IT departments and courses. We have a plan to collaborate with enthusiastic responders of this survey to develop effective project team.

Acknowledgment

This research is supported by JSPS KAKENHI Grant Numbers 16K01022 as well as by the Ministry of Education, Culture, Sports, Science and Technology, Japan. The authors greatly appreciate the faculty members and the administration officers of the universities who took time to respond to our survey.

References

- Al-Ansari, H., Yousef, N. (2002). Coverage of competencies in the curriculum of information studies: An international perspective 1. *Education for information*, 20(3–4), 199–215.
- Computing at School (2008). Available at <https://www.computingatschool.org.uk/>
- EC (2007). *e-Skills for the 21st Century: Fostering Competiveness, Growth and Jobs*. Commission of the European Communities
- Goldweber, M., et al. (2011). Enhancing the social issues components in our computing curriculum: computing for the social good. *ACM Inroads*, 2(1), 64–82.
- Hagiya, M. (2015). Defining informatics across Bun-kei and Ri-kei, *Journal of Information Processing*, 23(4), 525–530.
- Information Processing Society of Japan (2018). *Computing Curriculum Standard J17*. (in Japanese). Available at https://www.ipsj.or.jp/annai/committee/education/j07/curriculum_j17.html
- Kakeshita, T., Ohtsuki, M. (2011). A web-based survey system to analyze outcomes and requirements: a case for college level education and professional development in ICT. In: *Proc. 5-th Int. Conf. on Society, Cybernetics and Informatics (IMSCI 2011)*, 82–87.
- K-12 Computer Science Framework. Available at <https://k12cs.org/>
- Kakeshita, T. (2017). National survey of Japanese universities on IT education: overview of the entire project and preliminary analysis. In: *Proc. Int. Conf. on Computer Supported Education (CSEU 2017)*, 607–618.
- Kakeshita, T. (2018). National survey of Japanese universities on computing education: Analysis of departments majored in computing discipline, *Olympiads in Informatics*, 12, 69–84. DOI: 10.15388/oi.2018.06
- Kawamura, K. (2008). Computing curriculum standard J07: computing in general education, *IPJS Magazine*, 49(7), 768–774. (in Japanese)
- Marshall, L. (2012). A comparison of the core aspects of the ACM/IEEE Computer Science Curriculum 2013 Strawman report with the specified core of CC2001 and CS2008 Review. In: *Proc. Second Computer Science Education Research Conference*, 29–34.
- Ministry of Education, Culture, Sports, Science and Technology (MEXT, 2017). *FY2016 School Basic Survey*. (in Japanese)
- National Curriculum Survey. (n.d.) (Retrieved June 9, 2018). Available at <https://www.act.org/content/act/en/research/national-curriculum-survey.html>
- Ohtsuki, M., Kakeshita, T., Takasaki, M. (2017). National survey of Japanese universities on IT education: analysis of educational computer system. In: *Proc. 12-th Int. Conf. on Digital Information Management (ICDIM 2017)*, 98–103.
- Simon, R. M., et al. (2018). Language Choice in Introductory Programming Courses at Australasian and UK Universities. In: *Proc. 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18)*, 852–857.
- Sumi, K., Kakeshita, T. (2017). National survey of Japanese universities on computing education: Analysis of IT education for high school teacher license on IT. In: *Proc. 12-th Int. Conf. on Digital Information Management (ICDIM 2017)*. 87–92.
- Takahashi, N., Kakeshita, T. (2017). National survey of Japanese universities on IT education: analysis of informatics in general education. In: *Proc. 12-th Int. Conf. on Digital Information Management (ICDIM 2017)*, 104–109.
- TIMSS & PIRLS International Study Center. (n.d.) (Retrieved June 9, 2018), *TIMSS 2015 Assessment Frameworks*. Available at <https://timssandpirls.bc.edu/timss2015/frameworks.html>
- Urban-Lurain, M., Weinshank, D.J. (2000). Is there a role for programming in non-major computer science courses?. In: *Proc. 30th Annual Frontiers in Education Conference, Building on a Century of Progress in Engineering Education, T2B-7*.



T. Kakeshita is an associate professor at Computing Division, Saga University, Japan. He received his Ph.D. degree in Computer Science from Kyushu University, Japan in 1989. His major research interests include quantitative analysis of ICT education and ICT certification, and complexity analysis of database and software systems. **He received** an excellent educator award from Information Processing Society of Japan (IP SJ) in 2013. **He joined many activities such as IPSJ educational activity, Certified IT Professional Certificate (CITP), accreditation at Japan Accreditation Board for Engineering Education (JABEE) and ISO standard development (ISO/IEC JTC1/SC7/WG20).**



M. Ohtsuki is a senior lecturer at Computing Division, Saga University, Japan. She received her Ph.D. from Kyushu University in 1999. Her major research interests include computer aided ICT education, and software development methodologies including software testing. She is a committee member of JaSST (Japan Symposium on Software Testing) in Tokyo and is a commissioner at ASTER (Association of Software Test EngineerRing). She published several books about software development tools such as CVS, CppUnit etc.

Survey and Analysis of Computing Education at Japanese Universities: Informatics in General Education*

Tetsuro KAKESHITA¹, Naoko TAKAHASHI², Mika OHTSUKI¹

¹*Faculty of Science and Engineering, Saga University, 840-8502, Saga, Japan*

²*Faculty of Economics, Kokugakuin University, 150-8440, Tokyo, Japan*

e-mail: kake@is.saga-u.ac.jp, n.takahashi@kokugakuin.ac.jp, mika@is.saga-u.ac.jp

Abstract. We conducted the first nationwide survey of computing education at Japanese universities in 2016. In this paper, we report the survey result of informatics in general education for all students at a university or a faculty. The survey covers various aspects including program organization, quality and quantity of educational achievement, students, teaching staff and computing environment. 739 answers are collected from 530 universities in response to the survey. The answers cover 70.5% of the Japanese universities, and approximately 81.6% of the 649 universities that responded to the survey. The Information Processing Society of Japan (IPSJ) and the Japanese Ministry of Education (MEXT) will utilize the survey result to develop a new computing curriculum standard J17 and national policy of computing education respectively.

Keywords: informatics in general education, web-based survey and analysis, college level education, curriculum design, quality assurance in education.

1. Introduction

Computing education is essential at modern universities, since IT (Information Technology) is necessary to enhance ability of an individual and is expected as a powerful innovation driver through integration with various technologies (CS for ALL, n.d.; European Committee, 2018). There are four types of computing education in Japanese universities:

- A. Computing education at a department or a course majored in computing discipline.

* This paper is a revised and extended version of the following paper written by the same author. N. Takahashi, T. Kakeshita, “National Survey of Japanese Universities on Computing Education: Analysis of Informatics in General Education”, in Proc. 12-th International Conference on Digital Information Management (ICDIM 2017), 104-109, 2017.

- B. Computing education at a non-IT department or a course as a part of their major field of study.
- C. Informatics in general education for all university students typically at the first or second academic year.
- D. Computing education to obtain high school teacher license on computing subjects.

We conducted the first nationwide survey of Japanese universities on computing education in 2016 (Kakeshita, 2017). The survey is composed of four survey types A through D described above as well as the survey type E for educational computer system.

In this paper, we shall report and analyze the survey results regarding informatics in general education (Kawamura, *et al.*, 2015; Kawamura, *et al.*, 2016), i.e. survey type C. Informatics in general education is implemented as a common subject for all undergraduate students belonging to a university or a faculty in Japan. Although this survey is focused on Japanese universities, such type of general computing education is expected internationally (Informatics Europe and ACM Europe, 2013; Libeskind-Hadas, 2015). Therefore, our survey and analysis result will be of interest to a wide range of the readers outside of Japan.

Our survey on educational contents is based on J07-GEBOK** (Kawamura, 2008), introduced in Section 2.2, which is developed by Information Processing Society of Japan (IPSJ) as a guideline for college level informatics in general education. J07-GEBOK was developed without a detailed survey of college level informatics in general education. Our analysis is necessary to develop realistic curriculum guideline and accreditation criteria to improve informatics in general education at university.

IPSJ will utilize our survey result to develop various types of college level computing education guidelines including the new J17 curriculum standard (IPSJ, 2018). The Japanese Ministry of Education (MEXT) will utilize the survey result to improve the national policy of computing education in Japan.

2. Survey Plan

2.1. Survey Questions

The following is the list of questions for survey type C. As the reader can understand from the list, our survey covers various aspects of computing education by considering the Japanese standards for establishment of universities and our experience of accrediting computing programs in Japan:

- Name of university and/or faculty.
- Respondent standpoint.
- Program organization:
 - Day time, night or remote program.

**GEBOK – General Education Body of Knowledge

- Required number of credits for graduation.
- Number of subjects provided.
- Quality and quantity of educational achievement:
 - See Section 2.2 for detail.
- Enrolled students:
 - Regular academic years of the program.
 - Number of students.
- Teaching staff:
 - Number, educational background, current specialized field, tenure of faculty members.
 - Number and workload of support staff.
 - Number and workload of teaching assistant students.
- Computing environment:
 - Educational computer system.
 - Student's own PC and utilization at class.
 - Educational programming language.
- Other topics:
 - Strength and future plan of the program.
 - Utilization of IT certification and qualification.
 - Special remarks.

2.2. Survey of Quality and Quantity of Educational Achievements

The survey of quality and quantity of educational achievements is the core of our survey. We define six achievement levels for knowledge and skill represented in Table 1. These levels are used to describe quality of education.

Table 1
Knowledge and Skill Level Definition

Level	Knowledge Level Definition	Skill Level Definition
0	Not taught (unnecessary or already taught at general computing education)	
1	Not taught because of the time limitation or because the level of the contents is too high	Taught at class with simple exercise
2	Taught at class. Students know each term	Taught at class with some exercise. Students can perform the topic if detailed instruction is provided
3	Taught at class. Students can explain the meaning of each term	Taught at experiment with more complex exercise. Students can perform the topic with simplified instruction
4	Taught at class. Students can explain relationship and/or difference among related terms	Students perform combined research project containing the topic so that the students can autonomously perform the topic
5	Taught at class or graduation research project. Students can teach related domain or subject of the terms to others	Students perform combined research theme containing the topic. Students can teach how to perform the topic to others

We utilize J07-GEBOK (Kawamura, 2008) in order to define knowledge areas of the informatics in general education. J07-GEBOK is proposed by the Information Processing Society of Japan (IPSJ) as a common body of knowledge for informatics in general education. The following is the list of areas of J07-GEBOK. Each area contains several learning units:

- Information and Communication.
- Digitalization of the Information.
- Computing Elements and Structure.
- Algorithms and Programming.
- Data Modeling and Operation.
- Information Network.
- Information Systems.
- Information Ethics and Security.
- Computer Literacy.

J07-GEBOK is a subset of the common BOK utilized for other survey types A, B and D. This is because the subjects assigned for informatics in general education are quite limited due to the restriction of teaching staff and the number of students learning the subjects.

A university or a faculty answers expected knowledge and skill levels of the students at each area of the BOK. At the same time, the organization answers the total number of students taking the subjects taught in each area.

As a result, quality and quantity of education at the organization is summarized using J07-GEBOK.

2.3. Survey Process

We prepared the survey in October 2016. We defined the survey questions and set up the web-based survey system (Kakeshita and Ohtsuki, 2011). We utilized the web-based survey since we did not exactly know the actual organization for this survey in advance. After preparing various documents such as user manual and detailed instruction of the survey questions, we sent the formal request letter to all universities in Japan with a reference letter from the Japanese Ministry of Education in order to increase the response rate.

The survey was executed for two months starting at the beginning of November 2016. Each survey responder must first register to the web system and then answer the questions listed in Section 2.1. We also provide FAQ and independent answers for the questions from the responders.

After closing the survey, we reviewed the collected answers and requested the responders for possible correction of the incomplete answers.

3. Overview of Informatics in General Education

3.1. Response Rate Analysis

We collected 739 answers from 69 national universities, 58 public universities, and 404 private universities in response to the survey type C. 447 registrations are from entire universities and 292 registrations are from faculties or campuses of a university. The number of responded universities are 531, corresponding to 71.8% of the universities in Japan. This demonstrates the reliability of our survey.

The number of universities responded to at least one survey type A–E is 649. This implies that 81.8% of the responded universities provide informatics in general education and this type of computing education is widely executed in Japanese universities.

3.2. Respondent Standpoint

We asked the survey respondents about their position within the university. We made the question to clarify whether they are secretariat staff or faculty members. 69.9% of the respondents were secretariat staff. Since informatics in general education is administrated by a university or a faculty, a secretariat staff may have answered the questions on behalf of the faculty members in charge. Other respondents are university officials, representatives of common education, and representatives of educational computer center. It is commonly observed at Japanese universities that representatives of common education belong to another faculty and the secretariat staffs are usually working on administration of common education as a delegate of the representative.

4. Program Organization

In the class formats of the subjects offered as required credits of informatics in general education, most of them are provided as lectures. The second choice is an exercise, followed by training, practice, and laboratory work. The number of classes is distributed from 0 to more than 100 at large-scale universities. Here, we report the cases of lecture and exercise.

There are 263 answers, or 35.5% of the responses, stated that they have no required credits for informatics in general education. Many of such universities provide computing education as elective subjects. This indicates that 64.5% of the Japanese universities have required credits for informatics in general education.

Such information provides realistic restrictions to develop a curriculum guideline for informatics in general education.

4.1. Lecture Courses (Required)

For the number of required credits of the subjects provided as a lecture, 63.3% of the answers are 0, while 16.6% of the answers are 2 credits. There was a computing department that responded with a maximum value of 18 credits. We also have medical universities that answered with the number of lecture hours instead of the number of credits. In this case, we converted the number of hours to the number of credits since 1 credit corresponds to 11.25 lecture hours.

For the total number of required lectures, 39.5% of the answers was 0, 22.7% of the answers was 1. The maximum value of required subjects was 50 from a large-scale comprehensive private university. Fig. 1. represents the number of responses to each answer excluding 0.

4.2. Exercise Subjects (Required)

For the number of required credits for the exercises, 64.8% of the answers was 0, 15% of the answers was 2 credits. This can be interpreted that teaching of an exercise require extensive student guidance so that it is more difficult for a university to provide computing exercise to all their students. The maximum value was 14 credits at a university majored in social science with 800 first-year students. On the other hand, for the number of required seminar subjects, 15% of the answers were for 1 and 2 subjects respectively. The maximum value was 44 credits at a university majored in health care with 100 first-year students.

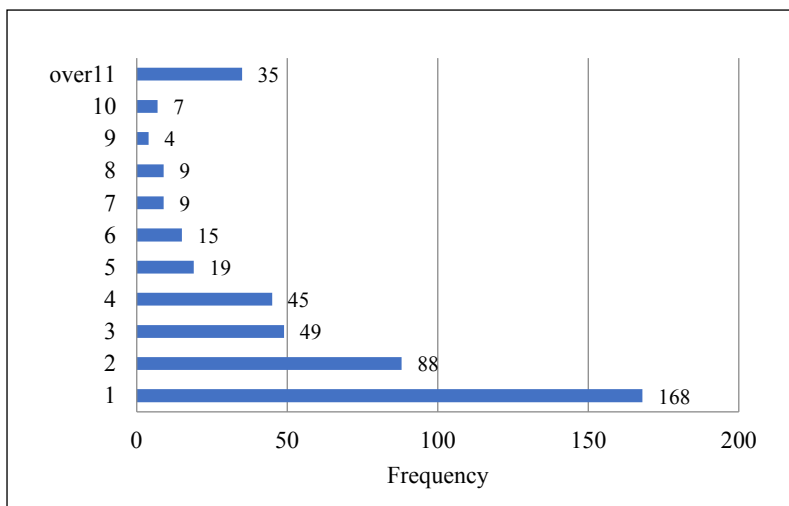


Fig. 1. Total Number of Lectures.

4.3. Number of Credits of Elective Subjects

For the number of credits required for graduation for elective computing subjects, 61.3% of the answers was 0 credits, and 11.8% of the answers was 2 credits. The answer includes all subject formats such as lecture, exercise, training, practice and laboratory work. Like the number of credits for the required subjects, the most popular answer was 2 credits.

5. Quality and Quantity of Educational Achievement

5.1. Overview of the Survey Areas

In the investigation of educational content, we asked universities to respond the expected level and number of courses based on the areas of J07-GEBOK. At the same time, we allowed them to select items from the reference standard of informatics (Hagiya, 2015) which defines the contents of computing education in Japan. Organization of the reference standard is summarized in Table 2. The reference standard is composed of 6 sections, 19 domains and 81 areas. The numbers within the parenthesis in Table 2 are the number of areas belonging to the section or the domain. Since J07-GEBOK is a small subset of the reference standard of informatics, we expected that very few universities teach contents of the reference standard. However, we found that all the areas are taught at some universities through the survey.

Table 2
Organization of the Reference Standard of Informatics

Section	Domain
(A) General Principles of Information (6)	
(B) Principles of Information Processing by Computers	Information Transformation and Transmission (4), Information Representation, Accumulation and Management (4), Information Recognition and Analysis (4), Computation (6), Algorithms (8)
(C) Technologies for Constructing Computers that Process Information	Computer Hardware (3), I/O Device (4), Fundamental Software (3)
(D) Understanding Humans and Societies that Process Information	Process and Mechanism for Information Creation and Transmission (2), Human Characteristics and Social System (3), Economic System and Information (2), IT-based Culture (2), Transition from Modern Society to Post Modern Society (2)
(E) Technologies and Organizations for Constructing and Operating “Systems” that Process Information in Societies	Technics for Information System Development (7), Technics to Obtain Information System Effect (6), Social System Related to Information (2), Principle and Design Methodology for HCI (4)
Competence	Professional Competency for IT Students (3), Generic Skill for IT Students (6)

We adopted the same definition of levels, as illustrated in Table 1, and BOK, illustrated in Table 2 along with J07-GEBOOK, throughout the survey types A to D in order to enable mutual comparison of the different survey types. Such comparison is important to understand relationship among different survey types.

There were 253 responses regarding the investigation of educational content and levels, which corresponds to 34.2% of the responses.

The universities are mainly focused on the areas in J07-GEBOOK, but the second most focused domain is “Generic Skill for IT Students” defined in Table 2. The generic skill contains creativity, logical and computational thinking, problem discovery and solving, communication and presentation, team work and leadership, and self-learning. It is well recognized that computing education is suitable to learn such generic skill.

5.2. Effort Analysis at Each GEBOOK Area

As for the areas of J07-GEBOOK, the degree that each university is focusing on is defined using the effort value. The effort value of an area is defined by the multiplication of the number of students learning the area and the average level of the students. We thus define two types of effort values corresponding to knowledge and skill.

Fig. 2 represents the effort values at each area of J07-GEBOOK. The analysis is useful to clarify the current effort distribution of the universities for the areas. The areas are sorted in descending order of the knowledge effort values. Even if the values are arranged in terms of knowledge and skill in the same order, computer literacy is ranked at the first place. The effort for the “data modeling and operation” is low. This is the same result as our previous investigation (Kawamura, 2015). We also observe that the skill effort is generally lower than the knowledge effort. The reason can be considered that teaching skill needs more effort than teaching knowledge since exercise becomes necessary.

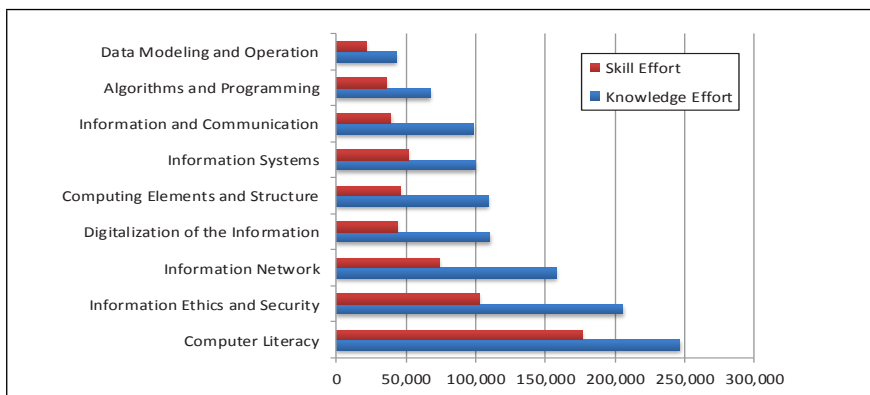


Fig. 2. Effort Values of Each Area of Informatics in General Education.

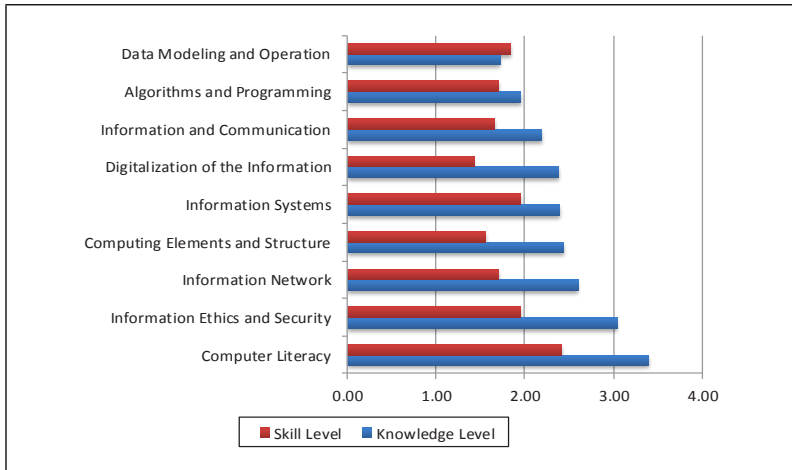


Fig. 3. Average Level of Each Area of Informatics in General Education.

5.3. Average Level at Each GEBOK Area

Fig. 3 represents average knowledge/skill levels at each GEBOK area. By this, the achievement levels can be analyzed in the current informatics in general education. Taking this into consideration, the requirement level can be defined at each area of general information education for the new curriculum recommendation. In addition to this, effort distributions of knowledge and skill in each GEBOK area can be utilized as a measure of the number of credits (or lecture hours) to be assigned to the areas.

A realistic curriculum can be designed by assigning appropriate number of hours and requirement level for each area considering these analyses' result.

5.4. Answer Distribution of Each GEBOK Area

Fig. 4–Fig. 12 represent the distributions of the number of responses at each GEBOK area. The readers can refer to Table 1 for the definition of knowledge and skill levels.

The responses of “Information and communication”, “Digitization of information”, “Elements and composition of computing”, “Information Network” and “Information Systems” have a similar trend. Skill is “not taught” in most of the answers, but knowledge level is separated at level 0 (not taught) and level 2. At level 1, there is more skill than knowledge (Fig. 4–Fig. 6, Fig. 9, Fig. 10). The “Algorithm and programming” and “Data modeling and operation” resulted in many level 0 (not taught) responses, which are different from other areas (Fig. 7, Fig. 8).

On the other hand, for “Computer literacy” (Fig. 12), the most frequent response for skill was level 2, and level 4 for knowledge. The most frequent response was level 1 for skill, and level 3 for knowledge in case of “Information Ethics and Security” (Fig. 11).

Considering these results, we made the following revision to develop J17-GEBOOK in J17 (IPJS, 2018). For “Data modeling and operation” with the smallest effort, we decided to change the area name to “Database and data modeling” and treat it mainly in the database. We changed the name of “Information system” to “Society and information system” because of strong relationship with society.

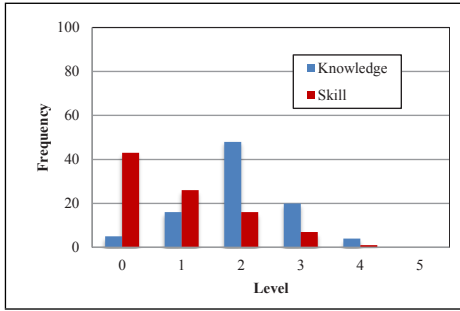


Fig. 4. Answer Distribution: Information and Communication.

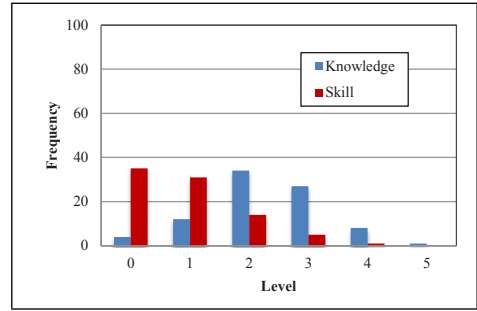


Fig. 5. Answer Distribution: Digitalization of the Information.

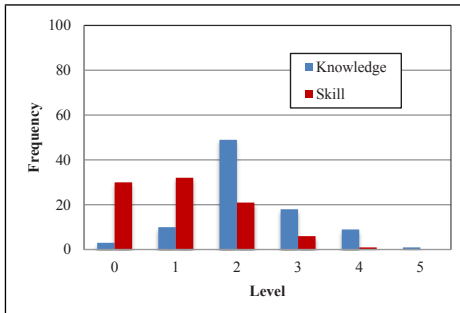


Fig. 6 Answer Distribution: Computing Elements and Structure.

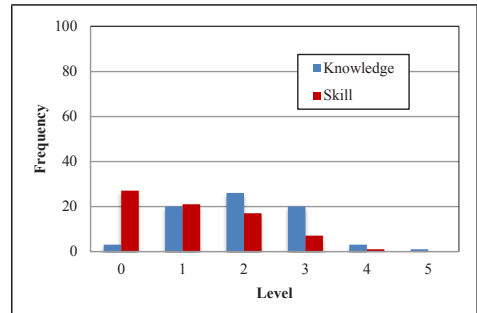


Fig. 7. Answer Distribution: Algorithm and Programming.

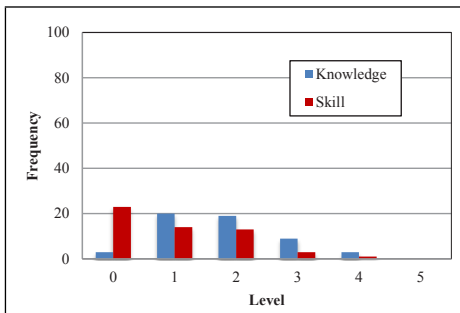


Fig. 8. Answer Distribution: Data Modeling and Operation.

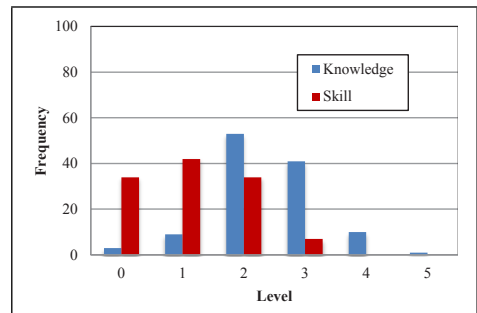


Fig. 9. Answer Distribution: Information Network.

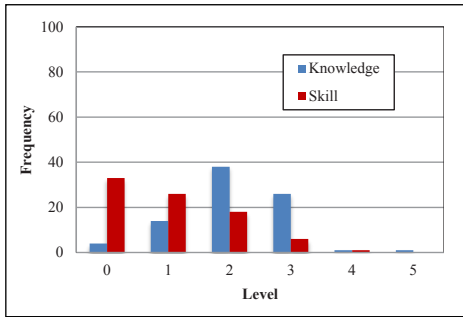


Fig. 10. Answer Distribution: Information Systems.

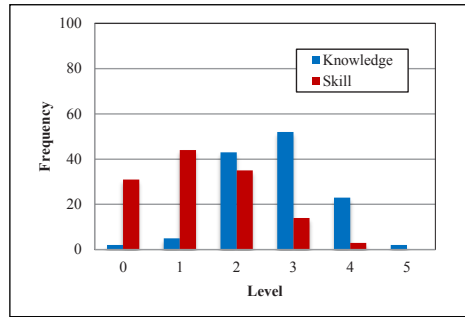


Fig. 11. Answer Distribution: Information Ethics and Security.

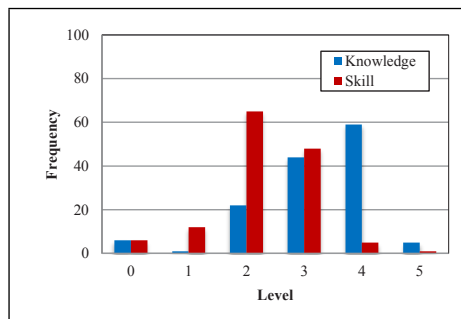


Fig. 12. Answer Distribution: Computer Literacy.

“Computer literacy”, which contains basic computer operation and application operation, is deemed to be handled in K-12 education at primary and/or secondary school so that it was dealt with as supplementary (pre-requisite) in J07. However, because the devoted effort to computer literacy is the largest, we decided to define an area called “Academic ICT Literacy” in J17 as the ICT skills to be handled at the higher education stage in combination with liberal arts education in a wide range of fields.

6. Status of Students and Faculty Members

6.1. Standard Target Students and Number of Courses

For the program’s standard target students, 284 (38.5% of the responses) programs are provided for the first-year students, while 334 (45.3%) programs are provided for the first and second-year students. However, we have 15% of the responses that included target students over third-year students. There is a need to assess whether informatics in general education is required for specialized courses and whether they should be mastered by fourth-year students.

The capacity of the first-year students at the universities responded to the survey is 416,062 as a total. The student capacity is between 100 and 500 at 49.5% of the responded universities, which is the most frequent answers.

The total number of male students was 137,633, while the total number of female students was 109,479. The total number of students was 247,112, which is equivalent to 59.4% of the capacity of the first-year students. We estimate that 247,000 students, approximately half of the first-year students estimated from the school basic survey (MEXT, 2016), learn informatics in general education, which indicates the importance of informatics in general education.

6.2. Status of Faculty Members in Charge

The general situation of the faculty members teaching informatics in general education courses is depicted in Table 3. Their current majors are determined by the faculty members based on whether their major field is included in the area “Computing” of the Grants-in-aid for Scientific Research.

Compared with the faculty members belonging to computing departments (Kakeshita, 2018), the ratio of full-time faculty members who graduated from a computing department and whose current major is computing is low. On the other hand, the ratio of employees who graduated from computing departments and whose current major is computing is higher for the case of part-time faculty members outside of the university. We also observe that the ratio of part-time faculty members outside of the university is 32% among all classifications of faculty members. This implies the shortage of full-time faculty members majored in computing discipline within the university.

Like the specialized computing education, faculty members with specialized knowledge of computing education should also be deployed for the faculty members in charge of informatics in general education.

We believe that although there are difficulties from the university’s side, but improvement is desirable in the future.

Table 3
Faculty Members Teaching Informatics in General Education

Category of Faculty Members	Total	Faculty Members who Graduated a Computing Department	Faculty Members whose Current Major is Computing
Full-time Faculty Members with Tennure	2,467	550 (24.1%)	318 (13.4%)
Full-time Faculty Members without Tennure	361	77 (21.3%)	130 (41.1%)
Part-time Faculty Members belonging to other Department	1,247	282 (22.6%)	443 (35.5%)
Part-time Faculty Members outside of the University	1,874	567 (30.6%)	891 (48.2%)
Total	5,849	1,476	1,782

6.3. Committee in Charge

The situation of the committee in charge of the administration of the general education is shown in Table 4. As a common education at the university or faculty level, we asked for the existence of a committee to oversee general computing education. 40.6% established a formal committee based on the campus regulations. However, 54.1% responded that organizations such as committees do not exist particularly. In our previous survey, about 60% responded that they have an administrative committee so that we have a similar result.

7. Educational Environment

7.1. Educational Computer System and Student PC Utilization

We asked for answers regarding educational computer systems which can be used for informatics in general education. 554 (74.9%) responded that there were PCs that could be used for university courses, while 173 (23.4%) responded that there was no PCs prepared that could be used for university courses mainly due to the shortage of financial support.

Table 5 represents PC utilization status possessed by the students. 83.2% of the universities answered that purchasing/owning a PC was voluntary for a student, while 6.8% answered that a student is required to purchase PC at the university level. Al-

Table 4
Committee in Charge

	# of Answers
Decisions made at informal meetings etc.	39
Established formal committee based on school regulations	301
None	400

Table 5
Utilization of Student PC

Utilization	# of Answers
All Students of the University must have PC	50
All Students of the Faculty must have PC	24
All Students of the Department/Course must have PC	13
Students are recommended to purchase PC	37
Purchasing of Student's own PC is optional	616
Total	740

though the most frequent case is purchasing of student's own PC is optional, many of the students are willing to purchase their own PC when they enter the university. This is because many educational contents are provided online, and students often prepare and submit various materials such as homework and job hunting application using their own PC.

7.2. LMS

Learning management system (LMS) is utilized at many universities in order to automate various educational activities such as report submission, online testing, student survey etc.

For the LMS utilization status, 47.6% responded that they did not utilize LMS, while 26.9% answered that they utilize an LMS based on commercial products. Regardless of whether the teachers actively utilize online submissions, there were approximately 100 responses which said that LMS was not used. Although further investigation is expected for the specific reason of this, we guess that individual faculty members accept student report via e-mail.

We obtained 290 responses, 75% of the cases which utilize LMS, that stated the product names. Table 6 summarizes the result of the LMS product names having more than 10 votes. Moodle occupies majority of the responses. We also received approximately 10 responses that two types of LMS are combined and used together.

7.3. Educational Programming Languages

We collected five educational programming languages from each university or faculty for which the students' achievement level is high. Table 7 illustrates popular programming languages for the informatics in general education calculated using the collected data. The score of each language is evaluated as a weighted sum of the answers. The

Table 6
Popular LMS Product

LMS Product Name	# of Answers
Moodle	92
WebClass	27
manaba	25
Blackboard	19
Course Power	13
Universal Passport	10

Table 7
Popular Programming Languages for Informatics in General Education

Language			Language		
	Language	Score		Language	Score
1	C	243	6	Ruby	48
2	Visual Basic/VBA	209	7	Fortran	35
3	Java	178	8	SQL	33
4	JavaScript	126	9	Python	25
5	C++	55	10	PHP	24

weight of a language is estimated using the rank, between 1 (5-th place) and 5 (1-st place), supplied by the university or the faculty. Although C language is the most popular as in the case of computing department, the second most popular language, Visual Basic, is different from the case of computing department.

7.4. Utilization of IT-related Certification

We obtained 59 responses, which was equivalent to 8% of the total responses. We found 13 IT certifications among the responses having two or more responses. They are depicted in Table 8. IT Passport Examination (IPA, n.d.), which covers a common and basic knowledge for utilizing IT, is the most popular examination and its share is 34.2% of the responses while Microsoft Office Specialist occupies 19.1%. This result was as expected since IT Passport Examination is authorized by the Ministry of Economy, Trade and Industry of Japan.

There were 14 responses that encouraged the acquisition of the IT certification since they are useful for job hunting for the students.

Table 8
Utilization of IT Certification

Qualification Name	# of Answers
IT Passport Examination (ITEE) by a Japanese government agency IPA	25
Microsoft Office Specialist (MOS)	14
Nissyo PC qualifying examinations by Japan Chamber of Commerce and Industry	5
.com Master	2
ICT Proficiency	2
Information Security Management	2

8. Concluding Remark

We can observe the entire picture of the computing education at Japanese universities through the survey. Although several problems are discovered, IPSJ is willing to improve the current situation through development of new computing curriculum standard J17 and cooperation with Ministry of Education, Japan.

Among the universities that responded to our survey, 530 universities (81.6% of the responded universities) provide informatics in general education, and 247,000 students (approximately half of the first-year students) are learning the course. This showed the importance of informatics in general education in Japan. However, while 64.5% responded that more than 1 credit is assigned, 87.6% responded that courses were offered with more than 1 subject, showing a discrepancy in the responses. There is a need to verify the cause of the difference in responses in the number of credits and subjects. Also, the knowledge and skill required for the exercise was designated at level 5, and there were universities with 50 subjects for the informatics in general education, and we obtained responses that we did not expect. We would like to clarify the meaning of these responses with additional investigations.

In the effort analysis including the reference standard for informatics, the second most common educational contents are “Generic Skill for IT Students”. We are planning to investigate relationship between computing education and generic skill training as a future research.

This survey was conducted using different methods than the ones used in our previous surveys during 2013 and 2014 (Kawamura, 2015). Thus, we cannot compare in a simple manner. Although equivalent results are obtained for some topics, such as the implementation rates of informatics in general education, there are significant differences in the position of the respondents and usage rate of LMS. As for the result of the educational content, the definitions of the knowledge and skill levels are different. Thus, there is a need to compare two survey result for reasonable interpretation of the difference.

IPSJ typically revises computing curriculum standard every 10 years. Conduction of a similar survey is expected every 5 years in order to observe the current status of computing education and to prepare the next curriculum standard to improve computing education.

Acknowledgment

This research is supported by JSPS KAKENHI Grant Numbers 16K01022 and 17K01036 as well as by the Ministry of Education, Culture, Sports, Science and Technology (MEXT), Japan. The authors greatly appreciate the faculty members and the administration officers of the universities who took time to respond to the survey.

References

- CS for ALL (n.d.). Available at <https://www.csforall.org/>
- European Commission (2018). Communication on the digital education action plan. Available at <https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:52018DC0022&from=EN>
- Hagiya, M. (2015). Defining informatics across Bun-kei and Ri-kei. *Journal of Information Processing*, 23(4), 525–530.
- Informatics Europe and ACM Europe (2013). Informatics education: Europe cannot afford to miss the boat. Available at <http://www.informatics-europe.org/images/documents/informatics-education-acm-ie.pdf>
- IPSJ (2018). *Computing Curriculum Standard J17*. Information Processing Society of Japan. (In Japanese). Available at https://www.ipsj.or.jp/annai/committee/education/j07/curriculum_j17.html
- IPA. (n.d.) *Japan's Information Technology Engineers Examination*. Information-technology Promotion Agency (IPA). Available at <https://www.ipa.go.jp/english/humandev/reference.html>
- Kakeshita, T., Ohtsuki, M. (2011). A web-based survey system to analyze outcomes and requirements: a case for college level education and professional development in ICT. In: *Proc. 5-th Int. Conf. on Society, Cybernetics and Informatics (IMSCI 2011)*, 82–87.
- Kakeshita, T. (2017). National survey of Japanese universities on IT education: overview of the entire project and preliminary analysis. In: *Proc. Int. Conf. on Computer Supported Education (CSEDU 2017)*, 607–618.
- Kakeshita, T. (2018). National survey of Japanese universities on computing education: Analysis of departments majored in computing discipline. *Olympiads in Informatics*, 12, 69–84.
- Kawamura, K. (2008). “Computing Curriculum Standard J07: Computing in General Education (J07-GE)”, *IPSJ Magazine*, Vol. 49, No. 7, 768–774. (in Japanese)
- Kawamura, K. *et al.* (2015). Research regarding the construction of Computing in General Education models in universities. JSPS KAKENHI Grant number 25350210. (In Japanese)
- Kawamura, K. *et al.* (2016). *Computing Education for Future University*. Nikkei BP Marketing. (In Japanese). Available at <https://sites.google.com/site/ipsj2010sigge/>
- Libeskind-Hadas, R. (2015). Every college student should take a computer science course. Available at https://www.huffingtonpost.com/ran-libeskindhadas/every-college-student-sho_b_7192700.html
- MEXT (2017). *FY2016 School Basic Survey*. Ministry of Education, Culture, Sports, Science and Technology of Japan. (in Japanese)



T. Kakeshita is an associate professor at Computing Division, Saga University, Japan. He received his Ph.D. degree in Computer Science from Kyushu University, Japan in 1989. His major research interests include quantitative analysis of ICT education and ICT certification, and complexity analysis of database and software systems. He received an excellent educator award from Information Processing Society of Japan (IPSJ) in 2013. He joined many activities such as IPSJ educational activity, Certified IT Professional Certificate (CITP), accreditation at Japan Accreditation Board for Engineering Education (JABEE) and ISO standard development (ISO/IEC JTC1/SC7/WG20).



N. Takahashi is a professor at Faculty of Economics, Kokugakuin University, Japan. She majored in mathematics at the university. After graduation, she worked at Fujitsu Ltd. as the first female SE. Next, she opened a PC school at an IT company. After independence, she worked on technical writing and taught PC skills and information systems at university. Since she worked at university, she specializes in computing education. She joined many activities such as IPSJ educational activity, **Committee of Informatics in General Education**, Committee of the entrance exam with an Informatics subject.



M. Ohtsuki is a senior lecturer at Computing Division, Saga University, Japan. She received her Ph.D. from Kyushu University in 1999. Her major research interests include computer aided ICT education, and software development methodologies including software testing. She is a committee member of JaSST (Japan Symposium on Software Testing) in Tokyo and is a commissioner at **ASTER (Association of Software Test EngineerRing)**. She published several books about software development tools such as CVS, CppUnit etc.

Constructionist Attempts at Supporting the Learning of Computer Programming: A Survey

Michael LODI¹, Dario MALCHIODI², Mattia MONGA²,
Anna MORPURGO², Bernadette SPIELER³

¹*Alma Mater Studiorum – Università di Bologna & INRIA Focus, Italy*

²*Università degli Studi di Milano, Italy*

³*Graz University of Technology, Austria*

*e-mail: michael.lodi@unibo.it, {malchiodi, monga, morpurgo}@di.unimi.it,
bernadette.spieler@ist.tugraz.at*

Abstract. Although programming is often seen as a key element of constructionist approaches, the research on learning to program through a constructionist strategy is somewhat limited, mostly focusing on how to bring the abstract and formal nature of programming languages into “concrete”, possibly tangible objects, graspable even by children with limited abstraction power. We survey the literature in programming education and analyse some programming languages designed to help novices from a constructionist perspective.

Keywords: programming, programming languages for learning, notional machine, constructionism.

Introduction.

While programming is often seen as a key element of constructionist¹ approaches (starting from LOGO (Feuerzeig *et al.*, 1970), a programming language designed to enable learning abstract concepts of disciplines like math, geometry, physics, and potentially all others, by manipulating computational objects (Papert, 1980)), the research on learning to program through a constructionist strategy is somewhat limited, mostly focusing on how to bring the abstract and formal nature of programming languages into “concrete” or even tangible objects, accessible also to children with lim-

¹ Constructionism originated from Seymour Papert, drawing on Jean Piaget’s constructivist view that knowledge needs to be (re)constructed rather than transmitted (Piaget, 1973), and adding that this is particularly effective when involves the construction of a (concrete or abstract) artifact, meaningful for the learner (Papert, 1980).

ited abstraction power (Resnick *et al.*, 2009; Kay *et al.*, 1997; Horn and Jacob 2007; Dann *et al.*, 2008; Hauswirth, Adamoli, and Azadmanesh, 2017). Notwithstanding this, programming is in some sense intrinsically constructionist, as it always involves the production of an artifact that can be shown and shared. Of course, this does not mean that programming automatically leads to constructivist/constructionist pedagogies: in fact, we see very different approaches, from open project-based learning to much more traditional education through lectures and closed exercises. Specific languages and environments play an important role too: for example, visual programming languages make it easier (by removing the request to face unnatural textual syntactic rules) to realize small but meaningful projects, keeping students motivated, and support a constructionist approach where students are encouraged to develop and share their projects – video games, animated stories, or simulations of simple real-world phenomena. Constructionist ideas are also floating around mainstream programming practice and they are even codified in some software engineering approaches: agile methods like eXtreme Programming (Beck and Andres, 2004), for example, suggest several techniques that can be easily connected to the constructionist word of advice about discussing, sharing, and productively collaborating to successfully build knowledge together (Resnick, 1996); moreover the incremental and iterative process of creative thinking and learning (Resnick, 2007) fits well with the agile preference to “responding to change over following a plan” (Beck *et al.*, 2001). It actually originated by observing how the traditional kindergarten approach to learning is ideally suited to learn to think creatively, and it is now called “creative learning spiral” (Fig. 1). According to this model, when one learns by creating something (*e.g.*, a computer program) she *imagines* what she wants to do, *creates* a project based on this idea, *plays* with her creation, *shares* her idea and her creation with others, *reflects* on the experience and feedback received from others, and all this leads her to *imagine* new ideas, new functionalities, new improvements for her project, or new projects. The process is iterated many times. This spiral describes an iterative process, highly overlapping with the iterative software development cycle.

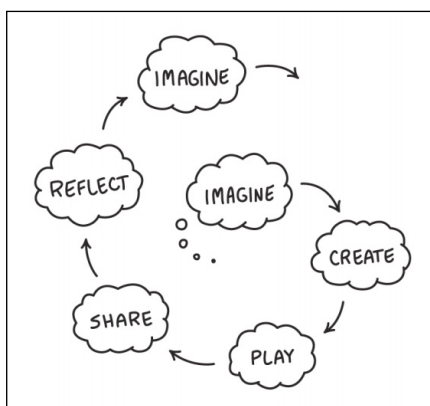


Fig. 1. Creative learning spiral (source: (Resnick, 2017)).

What does it mean to learn programming?

The basic premise behind programming – *i.e.*, producing a precise description of how to carry out a task or to solve a problem – is that an *interpreter*, different from the producer of the description, can understand it and effectively carry out the task as described. There are thus two distinct but tightly tied aspects in programming:

- i. the program itself (the text or other streams of symbols or actions that build up the digital coding of an algorithm);
- ii. the actions that take place when the program is run by the interpreter.

This distinction is explicit in most of the professional programming environments, but it is conceptually present even in those environments designed for very small children, where the program is somewhat implicit. The Bee-Bot², for example, is a bee-shaped robot that can be programmed by pushing the buttons on its back: the program, while recorded and then executed by the machine, is not explicit nor visible in its static form by the children, but it exists, and the programmer needs to master the relationship between the actions she records into the bee and the actions the bee will perform when the program will be executed. In this paper, however, we focus on programs in which the source code is explicit, as it is common in programming activities proposed to secondary school pupils.

Thus, one needs to know the interpreter in order to program, in particular:

- the set of basic actions it is able to perform;
- the language it is able to understand, with rules on how to compose basic actions;
- the relation between *syntax* and *semantics*, that is what actions it will perform given a description, and, conversely, how to describe a given sequence of actions so that it will perform them.

The first aspect, that is the program *source code*, is explicit, visible. The second one instead, that is the actions that take place when the program is run, is somewhat implicit, hidden in the execution time world, and not so immediate to grasp for novices. Moreover, this aspect is sometimes underestimated by both teachers and learners: teachers, as experts, give it for granted; learners tend to construct personal intuitive, not necessarily coherent, ideas of what will happen.

This dichotomy of programming – its static visible code and its implicit dynamics – emerges as a critical issue when learning to program, as shown by studies from different perspectives. In the following we cite a few (Sorva, 2013).

- Phenomenography studies show how novice programmers tend to perceive programming as no more than the production of code, missing to relating instructions in the program to what happens when the program is executed.
- Studies on programming misconceptions point out how most of programming misconceptions have to do with aspects that are not readily visible in the code but are related to the execution time, both in term of what will happen and of what will not unless explicitly specified in the code.

² <https://www.bee-bot.us/>

- Threshold concept theory identifies program dynamics as a candidate threshold concept in programming as it has many of the features that characterize threshold concepts; among others: it is a troublesome barrier to student understanding, it transforms how the student perceives the subject, it marks a boundary between programmers and end users.

To help novice programmers take into account also the dynamic side of programming, the concept of *notional machine* (Du Boulay, 1986; Sorva, 2013) has been proposed. A notional machine is a characterisation of the computer in its role as executor of programs in a particular language (or set of languages, or even a subset of a language) for didactic purposes. It thus gives a convenient description of the association syntax-semantics.

The following learning outcomes should therefore be considered when teaching to program:

- the development by students of a perception of programming that does not reduce to the production of code, but includes relating instructions to what will happen when the program is executed, and eventually comes to include producing applications for use and seeing it as a way to solve problems;
- the development of a mental model of a notional machine that allows them to make the association (static) syntax – (dynamic) semantics and to trace program execution correctly and coherently.

In particular, this latter outcome goal will include the development of some important skills.

- Given a program (typically one's own) and an observed behaviour:
 - identify when debugging is needed because the behaviour is somewhat not the one intended;
 - identify where a bug has occurred;
 - be able to correct the code.
- Given a program and its specification, be able to test it.
- Understand that there can be multiple correct ways to program a solution.

If these are crucial points in learning to write executable descriptions, however, programming is indeed a multifaceted competence, and the knowledge to construct and the skills to develop span over several dimensions, besides predicting concrete semantics of abstract descriptions. A skilled programmer needs to:

1. understand general properties of automatic interpreters able to manipulate digital information;
2. think about problems in a way suitable to automatic elaboration;
3. devise, analyze, compare solutions;
4. adapt solutions to emerging hurdles and needs;
5. integrate into teamwork and be able to elicit, organize, and share the abstract knowledge related to a software project.

Here we mainly focus on skill 1 and the support provided by programming languages and environments. Moreover we highlight the opportunity provided by agile methodologies to develop skill 5.

Unplugged Activities

Offline or *unplugged* programming activities have often been used to explain important concepts or vocabulary to students without actually using a PC, laptop, or smartphone, *e.g.*, x/y coordinates, the need for precise instructions for computers/robots, or variables and lists. Examples are to program a classmate like a robot, give paint instructions, pack a rucksack, or send “broadcast messages” to colleagues.

Unplugged activities in small groups have become popular over the years to introduce basic computer science concepts in non-vocational contexts, as they offer a number of interesting features.

- **A constructivist environment.**
 - Indeed by manipulating real objects or dramatising processes, pupils can observe what happens, formulate hypotheses, validate them through experiments, *i.e.* develop a scientific approach to the construction of their knowledge.
 - By working in a group, pupils are encouraged to participate, share ideas, verbalize and uphold their deductions.
- **Inexpensive set up:** they usually require very basic and inexpensive materials, so they can be easily proposed in different contexts.
- **No technological hurdles:** they allow students (and teachers) to have meaningful experiences related to important CS concepts (like algorithms) without having to wait until they get some technology and programming fluency (Bell and Lodi, to appear).

It is important to note that evidence shows unplugged activities should not replace programming activities, but can be helpful to make them more effective (Bell and Vahrenhold, 2018).

The following two examples, taken from CS Unplugged³ and ALaDDIn⁴, illustrate typical unplugged approaches to introduce children to programming.

In CS Unplugged “Rescue Mission”, pupils are given by the teacher a very simple language with only three commands: 1 step forward, 90 degrees left, 90 degrees right. The task is to compose a sequence of instructions to move a robot from one given cell on a grid to a given other cell. Pupils are divided into groups of three where each one has a role: either programmer, bot, or tester. This division of roles is done to emphasize the fact that programs cannot be adjusted on the fly; they must be first planned, then implemented, then tested and debugged until they work correctly.

ALaDDIn “Algomotricity and Mazes” is an activity designed according to a strategy called algomotricity (Lonati *et al.*, 2011; Bellettini *et al.*, 2012, 2013, 2014), where pupils are exposed to an informatic concept/process by playful activities which involve a mix of tangible and abstract object manipulations; they can investigate it firsthand, make hypotheses that can then be tested in a guided context during the activity, and eventually

³ <https://csunplugged.org/>

⁴ <http://aladdin.di.unimi.it/>

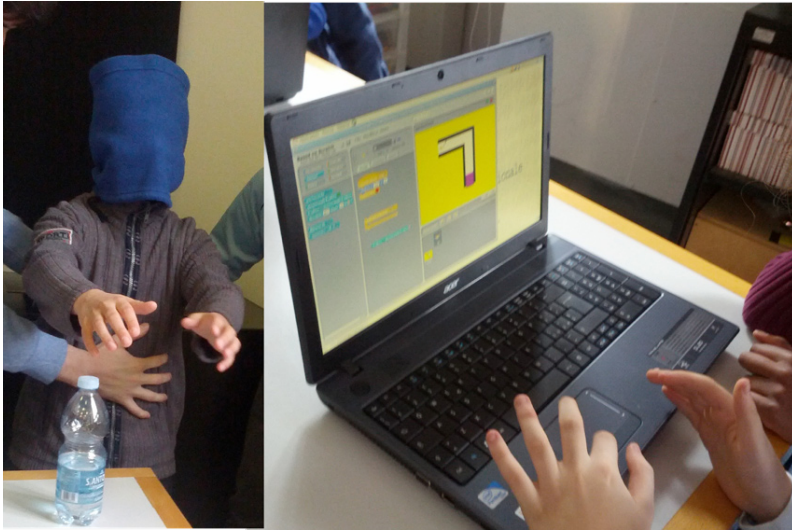


Fig. 2. The first and last phase of the “Algomotricity and Mazes” activity, respectively.

construct viable mental models. Algomotricity starts “unplugged” (Bell, Rosamond, and Casey, 2012) but ends with a computer-based phase to close the loop with pupils’ previous acquaintance with applications (Taub, Armoni, and Ben-Ari, 2012).

“Algomotricity and Mazes” focuses on primitives and control structures. The task is that of verbally guiding a “robot” (a blindfolded person) through a simple path. Working in groups, pupils are requested to propose a very limited set of primitives to be written each on a sticky note, and to compose them into a program to be executed by the “robot”. Also, they have the possibility of exploiting basic control structures (if, repeat-until, repeat-n-times). The conductor may decide to swap some programs and “robots”, in order to emphasize the ambiguity of some instructions or the dependency of programs on special features of the “robot” (e.g., step/foot size). In the last phase, students are given computers and a slightly modified version of Scratch. They are requested to write programs that guide a sprite through mazes of increasing complexity where shape patterns foster the use of loops.

Notional Machines

An important intuition for approaching programming from a constructionist perspective is that programs are a join point between our mind and the computer, the interpreter of the formal description of what we have in mind. Thus, programs appeal to our curiosity and ingenuity and are wonderful artifacts to share and discuss with other active minds. Such a sharing, however, assumes that the interpreter is a shared knowledge among peers. When a group of people programs the same ‘machine’, a shared *semantics* is in

fact given, but unfortunately people, especially novices, do not necessarily write their programs for the formal interpreter they use, rather for the *notional machine* (Sorva, 2013; Berry and Kölling, 2014) they actually have in their minds.

A notional machine is an abstract computer responsible for executing programs of a particular kind (Sorva, 2013) and its grasping refers to all the general properties of the machine that one is learning to control (Du Boulay, 1986). The purpose of a notional machine is to explain, to give intuitive meaning to the code a programmer writes. It normally encompasses an idealized version of the interpreter and other aspects of the development and run-time environment; moreover, it should bring also a complementary intuition of what the notional machine cannot do, at least without specific directions of the programmer.

To introduce a notional machine to the students is often the initial role of the instructors. Ideally this should be somewhat incremental in complexity, but not all programming languages are suitable for incremental models: in fact, most of the success for introductory courses of visual languages or Lisp dialects is that they allow shallow presentations of syntax, thus letting the learners focus on the more relevant parts of their notional machines.

An explicit reference to the notional machine can foster meta-cognition and, during teamwork, it can help in identifying misconceptions. But how can the notional machine be made explicit? Tracing of the computational process and visualization of the execution are effective candidate tools. They allow instructors to make as clear as possible: (i) what novice programmers should expect the notional machine will do and (ii) what it actually does.

Abstract Programming Patterns

A small number of abstract programming patterns can be applied to a potentially infinite spectrum of specific conditions. This is often a challenge for novices, given that most of the times the discipline is taught (i) introducing one or more primitive tools (e.g., variables), and (ii) showing some examples highlighting how these tools can be used to solve specific problems. This might lead to the rise of *misconceptions* of pupils w.r.t. the above-mentioned tools.

The concept of *role of variables* (Sajaniemi, 2002; Proulx, 2000) has been proposed in order to guide novice programmers from the operational knowledge of a variable as the holder of a mutable value to the ability to identify *abstract* use cases following a small number of roles (such as those in Fig. 3). Such ability is of great help when tackling the solution of a specific problem, for instance, that of computing the maximal value within a sequence. Indeed, this is a great opportunity for letting pupils realize that this problem is a special case of the more general quest for optimal value. The latter can be found using a *most-wanted holder* to be compared with each element of the sequence and containing the highest value seen so far. This method easily fits the search of the maximal as well as the minimal value, and it also efficiently handles less obvious cases such as that of finding the distinct vowels occurring in a sentence.

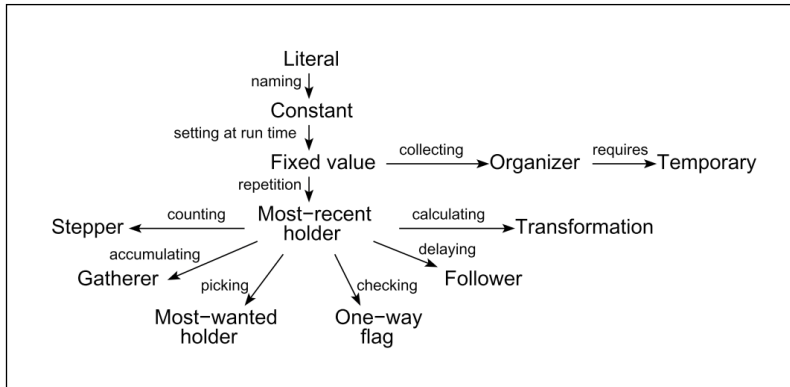


Fig. 3. Roles of variables, organized in a constructionist-like hierarchy where the predecessor of an arrow is a prerequisite for learning the corresponding successor (source: (Sajaniemi, 2002)).

These roles can also be gradually introduced following the hierarchy of Fig. 3, starting from the concept of literal (e.g., an integer value or a string) and building knowledge about one role on the top of already understood roles.

For selection and iteration as well there are several standard use patterns that occur over and over again. Selection patterns (Bergin, 1999) and loop patterns (Astrachan and Wallingford, 1998) have been introduced with the same goal. For instance, to illustrate the idea, the *loop and a half* pattern is an efficient processing strategy for a sequence of elements whose end can be detected only *after* at least one element has been read. It uses an infinite loop whose body accesses the next sequence element. If there are no more elements, the loop is escaped through a controlled jump, otherwise some special actions are possibly executed before continuing the iteration. The code snippet shown in Fig. 4 shows one of the canonical incarnations of this pattern: the possibly repeated check of a value given as input, detecting and ignoring invalid entries.

Selection and loop patterns fit well within a constructionist-based learning path: they might be *naturally* discovered when critically analyzing software implementations. For instance, the previous loop could be the end point of a reasoning scheme started from the detection of a duplicated line of code in a quick-and-dirty initial implementation.

```

while True:
    value = input('insert a positive, odd value')
    if value > 0 and value % 2 == 1:
        break
    print('the value is not valid')
  
```

Fig. 4. A typical loop and a half pattern applied to the repeated validation of external inputs to a procedure.

In general, abstract programming patterns are provided in a short number, in order to cover them within a standard introductory computer programming course; moreover, the related concepts are easily grasped by experienced computer science teachers (Ben-Ari and Sajaniemi, 2004), thus they can be embedded in already existing curricula with low effort.

Misconceptions

Sorva defines *misconceptions* as “*understandings that are deficient or inadequate for many practical programming contexts*” (Sorva, 2013).

Some authors (Ben-Ari, 2001) believe that computer science has an exceptional position in constructivist’s view of knowledge constructed by individuals or groups rather than a copy of an ontological reality: in fact, the computer forms an “accessible ontological reality” and programming *features many concepts that are precisely defined and implemented within technical systems [...] sometimes a novice programmer “doesn’t get” a concept or “gets it wrong” in a way that is not a harmless (or desirable) alternative interpretation. Incorrect and incomplete understandings of programming concepts result in unproductive programming behavior and dysfunctional programs* (Sorva, 2013).

According to Clancy, there are two macro-causes of misconceptions: *over- or under-generalizing* and *a confused computational model*. High-level languages provide an abstraction on control and data, making programming simpler and more powerful, but, by contrast, hiding details of the executor to the user, who can consequently find mysterious some constructs and behaviors (Clancy, 2004).

Much literature about misconceptions in CSEd can be found: we list some of the most important causes of misconceptions, experienced especially by novices, divided into different areas, found mainly in (Clancy, 2004; Sirkiä, 2012; Sorva, 2013) and in the works they reference. For a complete review see for example (Qian and Lehman, 2017).

English

Keywords of a language do not have the same meaning in English and programming. For example, the word *while* in English indicates a constantly active test, while the construct `while` can test the condition again only at the beginning of the next iteration. Some students believe that the loop ends at the precise moment the condition is falsified. Similarly, some of them think of the `if` construct as a test continuously active and awaiting the occurrence of a condition, others believed that the `then` branch is executed as soon as the condition becomes true.

Syntax

Although one may think the syntax is one of the biggest sources of misconceptions, studies show that it is a problem only in the very early stages. In particular, some students were able to write syntactically valid programs, which, however, were not useful for solving the given problem, or were semantically incorrect.

Mathematical notation

Reported by many authors, classical is the confusion that generates the assignment with the = symbol (for example, seen as an equation or as a swap of values between variables) or the increment ($a = a + 1$) thought of as an impossible equation.

Examples of over-generalization

Some authors found a series of non-existent constraints (e.g., methods in different classes that must have different names, arguments that can only be numbers, “dot” operator usable just in methods) dictated by the fact that the students had not seen any counterexample for such situations.

Similarities

The analogy “a variable is like a box” can foster the idea that – like a box – it can contain more elements at the same time. The analogy “programming with the computer is like conversing with it” can bring to attribute *intentionality* to the computer and therefore to think that it:

- has a hidden intelligence that understands the intentions of the programmer and helps her achieve her goal (the so-called “superbug”);
- has a general vision, knowing also what will happen in lines of code that it is not currently running.

Some aspects of programming are particular carriers of misconceptions.

Sequence

Many misconceptions are due to lack of understanding of the program flow: all lines active at the same time, “magic” parallelism, the unimportance of the order of instructions, difficulty in understanding the branches.

Passing parameters

Students present difficulties in this area, for example by confusing the types of passing (by value, by reference, ...), making mistakes with the return value or with the parameters’ scope.

Input

Input statements are particularly problematic. Students do not understand where the input data come from, how they are stored and made available to the program. Some of them believe that a program remembers all the values associated with a variable (its “history”).

Memory allocation

There are considerable difficulties in understanding the memory model of languages where allocation happens implicitly.

Programming Languages for Learning to Program

From a constructionist viewpoint of learning, programming languages have a major role: they are a key means for sharing artifacts and expressing one's theories of the world. The crucial part is that artifacts can *be executed* independently from the creator: someone's (coded) mental process can become part of the experience of others, and thus criticized, improved, or adapted to a new project. In fact, the origin of the notion itself of constructionism goes back to Papert's experiments with a programming environment (LOGO) designed exactly to let pupils tinker with math and geometry (Papert, 1980). Does this strategy work even when the learning objective is the programming activity itself? Can a generic programming language be used to give a concrete reification of the computational thinking of a novice programmer? Or do we need something specifically designed for this activity? Alan Kay says that programming languages can be categorized in two classes: "agglutination of features" or "crystallization of style" (Kay, 1993). What is more important for learning effectively in a constructivist way? Features or style?

In the last decade, a number of block-based programming tools have been introduced to help students have an easier time when first practicing programming. These tools, often based on web-based technologies, as well as an increase in the number of smartphones and tablets, opened up new ways for innovative coding concepts (Kahn, 2017). In general, they focus on younger learners, support novices in their first programming steps, can be used in informal learning situations, and provide a visual language which allows students to recognize blocks instead of recalling syntax (Tumlin, 2017). Many popular efforts for spreading computer science in schools, like (Goode, Chapman, and Margolis, 2012) or the teaching material from Code.org,⁵ rely on the use of such environments. In addition, such tools have been adopted into many computing classes all over the world (Meerbaum-Salant, Armoni, and Ben-Ari, 2010).

LOGO

LOGO was designed (since 1967) for (constructionist) educational purposes by Wally Feurzeig, Seymour Papert, Cynthia Solomon, Daniel Bobrow, and Richard Grant (Papert, 1980). Its syntax was heavily influenced by Lisp (at the time the standard language for Artificial Intelligence research) and it was initially designed to aid students in learning secondary school mathematics. The most successful LOGO version featured a graphical (at least in principle) environment: instructions are directed to a "turtle" who moves around the screen, possibly leaving a colored trace. The turtle should help learners (especially the younger ones) with a sort of self-identification: its movements have a clear correspondence with their movements in the real world. The patterns drawn by the turtle can be the way the learners build their understanding

⁵ <https://hourofcode.com>


```
TO CIRCLE
  REPEAT FOREVER
    [
      FORWARD 1
      RIGHT 1
    ]
```

Fig. 5. A procedure to draw a circle in LOGO.

of 2D geometry, discovering in the process even deep mathematical truths as the fact that a circle can be approximated by a high number of straight segments (Abelson and DiSessa, 1986) (see Fig. 5).

Interestingly enough, LOGO was originally conceived to empower learners of mathematics/geometry, not programming. Programming is *just* a means of expression, but one with great epistemic potential. According to Papert: “in teaching the computer how to think, children embark on an exploration about how they themselves think. The experience can be heady: Thinking about thinking turns every child into an epistemologist, an experience not even shared by most adults” (Papert, 1980). Also, by expressing something in a way the LOGO turtle can “understand” can be fruitful for real-world activities, too. Juggling, for example, can be analyzed with LOGO: the identification of *proper* sub-activities (*i.e.*, sub-routines like TOP-RIGHT to recognize when one juggling ball is at the top of its trajectory going to the right, or TOSS-LEFT to throw the ball with the left hand) may shorten significantly the time for acquiring juggling skills (from days to hours, according to (Papert, 1980)). And here ‘proper’ should be understood as appropriate to the task, but also as “fitting properly with the programming language idiomatic way of describing computational processes”. LOGO had many independent implementations and its approach is still very popular, even Python has a `turtle` package in its standard library.

Smalltalk

Smalltalk (Goldberg and Kay, 1976) also has its roots in constructionist learning. Back in the early seventies, at the Learning Research Group within the Xerox Parc Research Center, people were envisioning a world of personal computing devices which should have “programmability”. Smalltalk, whose lineage traces clearly to LOGO and Lisp, was designed with a general audience in mind, since everyone should be comfortable with programming and computing devices should become ubiquitous in learning environments “along the lines of Montessori and Bruner” (Kay, 1993). Thus, Smalltalk was not directed specifically to children and it has conquered a wide professional audience. In Smalltalk everything is an ‘object’ able to react to ‘messages’. It follows a highly consistent object-oriented approach and code can be factored out by inheritance and dynamic binding. Smalltalk introduces also the idea that everything in the system is programmable: such a dynamic environment encourages a trial-and-error approach. A spe-

cific Smalltalk system for children was designed later as an evolution of Squeak Smalltalk: E-toys (Kay *et al.*, 1997) provided a world of “sprites”, funny characters that can be moved (concurrently) around the screen by programming them in Smalltalk. E-toys then evolved in Scratch, where the programming part was replaced by visual blocks.

BASIC, Pascal

It seems legitimate to mention BASIC (Beginner’s All-purpose Symbolic Instruction Code (Kurtz, 1978)) in a paper on constructionism and programming: for years BASIC has been the elective language for personal projects and even before widespread Internet connectivity, several communities shared BASIC programs in Bulletin Board Systems and magazines. Its popularity among self-taught programmers, however, was due mainly to its availability on personal and home computing devices. Moreover, the language was typically implemented using an interpreter, thus naturally fostering the trial-and-error and incremental learning styles typical of a constructionist setting. A generation grown with BASIC still thinks it is a wonderful approach to get children hooked on programming (see for example (Brin, 2016)). However, many believe BASIC is not able to foster good abstractions and fear that BASIC programmers will bring bad habits to all their future computational activities.

In 1970 Niklaus Wirth published Pascal (Wirth, 1993), a small, efficient language intended to encourage sound programming practices using structured programming and data structuring. For about 25 years, Pascal (and its successors like TurboPascal or Modula-2) was the most popular choice for undergraduate courses and a whole generation of computer scientist learned to program through its discipline popularized by Wirth in his book “Algorithms + Data Structures = Programs”. Only Java had similar success in undergraduate courses. However, while Java popularity was (and is) influenced by trends in the software industry, Pascal was appealing mainly for its intrinsic discipline, which matched the academic sentiment of the time.

Scheme, Racket

Scheme (Abelson *et al.*, 1998) is a language originally aimed at bringing structured programming in the lands of Lisp (mainly by adding lexical scoping). The language has nowadays a wide and energetic community of users. Its importance in education, however, is chiefly related to a book, “Structure and Interpretation of Computer Programs” (SICP) (Abelson, Sussman, and Sussman, 1996), which had a tremendous impact on the practice of programming education. The book derived from a semester course taught at MIT. It has the peculiarity to present programming as a way of organizing thinking and problem solving. Every detail of the Scheme notional machine is worked out in the book: at the end, the reader should be able to understand the mechanics of a Scheme interpreter and to program one by herself (in Scheme). The book, which enjoyed widespread adoption, was originally directed to MIT undergraduates and it is certainly not

suitable either for children or even adults without a scientific background: examples are often taken from college-level mathematics and physics.

A spin-off of SICP explicitly directed to learning is Racket. Born as ‘PLT Scheme’, one of its strengths is the programming environment DrScheme (Findler *et al.*, 2002) (now DrRacket): it supports educational scaffolding, it suggests proper documentation, and it can use different *flavours* of the language, starting from a very basic one (Beginning Student Language, it includes only notation for function definitions, function applications, and conditional expressions) to multi-paradigm dialects. The DrRacket approach is supported by an online book “How to design programs” (HTDP) ⁶ and it has been adapted to other mainstream languages, like Java (Allen, Cartwright, and Stoler, 2002) and Python. The availability of different languages directed to the progression of learning should help in overcoming what the DrRacket proponents identify as “the crucial problem” in the interaction between the learner and the programming environment: beginners make mistakes before they know much of the language, but development tools yet diagnose these errors as if the programmer already knew the whole notional machine. Moreover, DrRacket has a minimal interface aimed at not confusing novices, with just two simple interactive panes: a definitions area, and an interactions area, which allows a programmer to ask for the evaluation of expressions that may refer to the definitions. Similarly to what happens in visual languages, Racket allows for direct manipulation of sprites, see an example in Fig. 6.

The authors of HTDP claim that “program design – but not programming – deserves the same role in a liberal arts education as mathematics and language skills.” They aim at systematically designed programs thanks to systematic thought, planning, and understanding from the very beginning, at every stage, and for every step. To this end, the HTDP approach is to present “design recipes”, supported by predefined scaffolding that should be iteratively refined to match the problem at hand. This is indeed very close to the idea of micropatterns discussed above.






```
(define (picture-of-rocket.v3 height)
  (cond
    [(<= height (- 60 (/ (image-height  ) 2))]
     (place-image  50 height
                  (empty-scene 100 60)))
    [(> height (- 60 (/ (image-height  ) 2))]
     (place-image  50 (- 60 (/ (image-height  ) 2))
                  (empty-scene 100 60)))]))
```

Fig. 6. Racket code for “landing a rocket”.

⁶ Current version: <http://www.htdp.org/2018-01-06/Book/index.html>

Scratch, Snap!, Alice, and others

EToys worlds with pre-defined – although programmable – objects, evolved in a generic environment in which everything can be defined in terms of ‘statement’ blocks. Scratch (Resnick *et al.*, 2009), originally written in Smalltalk, is the most popular and successful visual block-based programming environment. Launched in 2007 by the MIT Media Lab, the Scratch site has grown to more than 25 million registered members with over 29 million Scratch projects shared programs.

Unlike traditional programming languages, here graphical programming blocks are used that automatically snap together like Lego bricks when they make syntactical sense (Ford, 2009). In visual programming languages, a block represents a command or action and they are arranged in scripts. The composition of individual scripts equals the construction of an algorithm. The building blocks offer the possibility, *e.g.*, to animate different objects on a stage, thus defining their behavior.

The Scratch environment has some distinctive characteristics, according to its authors (Maloney *et al.*, 2010). Among the ones the authors highlight, some are particularly relevant in the constructionist approach:

Liveness

The code is constantly running and can be changed on the fly, immediately seeing the runtime effects of the change; this encourages users to tinker with the code.

No error messages

When you play with Lego bricks, they stack together or they don’t – the same happens in Scratch; program always run: syntax errors are prevented from the block shapes and connections, and also runtime errors are avoided by doing something “reasonable” (*e.g.*, in the case of an out-of-range value); this is particularly important not to frustrate kids and to keep them iterating and developing: “*A program that runs, even if it is not correct, feels closer to working than a program that does not run (or compile) at all*” (Maloney *et al.*, 2010).

Other characteristics are useful to help novices avoiding misconceptions that often arise when starting to learn to program.

Execution made visible

A glowing yellow border surrounds running scripts (in some versions each block is highlighted when it is executed); this is very helpful in program reading and debugging, and helps students form a correct mental model of the notional machine underlying the program execution.

Making data concrete

You can see in a variable box, automatically shown, its current value: again, this is helpful for making the underlying machine model visible.

Finally, other characteristics introduce important software engineering and development concepts.

Open source

Each shared project has a “see inside” button that brings you to the project source; you can read and edit the blocks to see what happens.

Remixing

If you edit someone else’s project, you create a remix: you are the author, but the system automatically gives credits to the original author (at any depth, keeping track of multiple remixes in a tree) and suggests you to explicitly declare what changes you made.

The main limitation of Scratch programs is that they do not scale well from the abstraction point of view: only since version 2 you can “make a new block” that is, a procedure with optional parameters. These blocks have no possibility to return a value (like a number or a boolean) and so can’t be nested inside other blocks, forcing you to modify global variables if needed.

Snap!⁷ (originally BYOB, Build Your Own Blocks) is an extended reimplementaion of Scratch with functions and continuations. These added capabilities make it suitable for a serious introduction to computer science for high school or college students: in fact, Snap! is used as the basis for an Advanced Placement CS course at Berkeley⁸.

The Scratch approach was also ported to mainstream programming languages: in Alice (Dann, Cooper, and Pausch, 2008) visual blocks are in fact Java instructions. Alice worlds are 3D: this choice makes it very attractive and appealing to pupils (Rodger *et al.*, 2009), who can program amazing 3D animations. It also adds many complexities, since moving objects in a 3D space is not trivial.

Recently, these environments evolved towards web or phone/tablet versions, in order to be available in the contexts more popular within young people. For example, Pocket

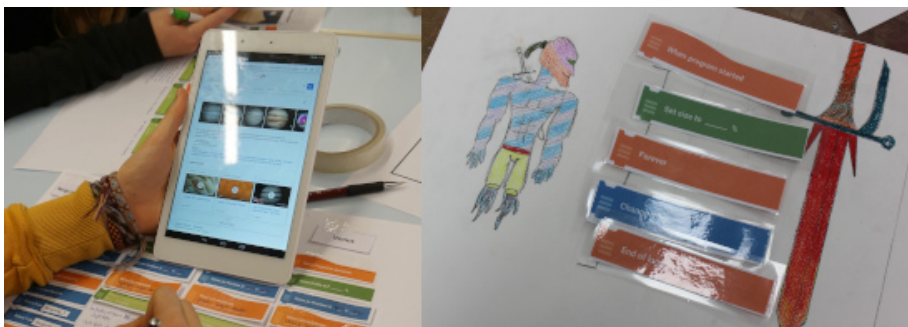


Fig. 7. Students design a program to be run with Pocket Code.

⁷ <https://snap.berkeley.edu/>

⁸ <https://bjc.berkeley.edu/>

Code⁹ allows the creation of games, stories, animations, and many types of other apps directly on phones or tablets, thereby teaching fundamental programming skills (Slany, 2014). In some cases block and textual programming languages are interchangeable. In many cases these environments can connect to physical devices and sensors, with the goal of increasing the constructionist appeal of block programming, and opening to the world of “tinkering” with electronics.

All in all, visual programming languages seem to provide an easier start and a more engaging experience for learners. The ease of use, simplicity, and desirability of new visual programming environments enables young people to imagine complex goals. A study which compared three classes that used either block-based (Scratch), text-based (Java), or hybrid blocks/text (Snap!/JavaScript) programming languages showed that students generally found block-based programming to be easier than the text-based environments (Weintrop and Wilensky, 2015). Some researchers, however, argue that students are not fully convinced that a visual language can help them learn other programming languages (Lewis *et al.*, 2014).

Common Features

The above short survey of programming languages for education shows they have some recurrent traits that link them to the themes discussed in the section “What does it mean to learn programming.”

Personification

The interpreter becomes a “persona”, computation is then carried out through anthropomorphic (or, better, zoomorphic, since animals are very common) actions. This seems to contradict a famous piece of advice coming from no less than E. W. Dijkstra (Dijkstra, 1985). Speaking of anthropomorphism in computer science, he noted: “The trouble with the metaphor is, firstly, that it invites you to identify yourself with the computational processes going on in system components and, secondly, that we see ourselves as existing in time. Consequently, the use of the metaphor forces one to what we call ‘operational reasoning’, that is reasoning in terms of the computational processes that could take place. From a methodological point of view, this is a well-identified and well-documented mistake: it induces a combinatorial explosion of the number of cases to consider and designs thus conceived are as a result full of bugs.” The *reasoning in terms of the computational processes*, however, is what is probably needed for a novice in order to familiarize with the notional machine.

Visualization and tracking

Computational processes that evolve in time are described by static texts: the mapping between the two is not trivial and it requires an understanding of the notional machine. Educational programming environments often try to make the mapping more explicit

⁹ <https://catrobat.org>

with some visualization of the ongoing process: the trace left by the LOGO turtle, or some other exposition of the changing state of the interpreter.

Appeal

Engagement of learners is crucial: to this end, it is important to give learners powerful libraries and building blocks. It is not clear, however, how to properly balance amazing effects in order to avoid they become a major distraction: sometimes children may spend their (limited) time in changing the colors of the sprites, instead of trying to solve problems.

Learning to Program in Teams

Constructivist approaches often emphasize the importance of social context in which the learning happens (see e.g. (Vygotsky, 1978)).

Working in developers teams requires new skills, especially because software products (even the ones in the reach of novices) are often tangled with many dependencies and division of labour is hard: it inevitably requires appropriate communication and coordination. Therefore, it is important that novice programmers learn to program in an “organized” way, discovering that as a group they are able to solve more challenging and open-ended problems, maybe with interdisciplinary contributions.

To this end, agile methodologies fit well with constructivist pedagogies involving learning in teams, and they are increasingly exploited in educational settings (see for example (Kastl, Kiesmüller, and Romeike, 2016; Missiroli, Russo, and Ciancarini, 2016)).

- Agile teams are typically small groups of 4–8 co-workers.
- Agile values (Beck *et al.*, 2001) (individuals and interactions over processes and tools; customer collaboration over contract negotiation; responding to change over following a plan; working software over comprehensive documentation) relate well with constructivist philosophies.
- Agile teams are self-organizing, emphasize the need for reflecting regularly on how to become more effective, and tune and adjust their behavior accordingly.
- Agile techniques like pair programming, test driven development, iterative software development, continuous integration are very attractive for a learning context.

The iterative nature of agile methods is well exemplified by test-driven development, or TDD (Beck, 2003). This technique reverses the order between code implementation and correctness test. Namely, the specification of the programming task at hand is actually provided with a test the *defines* correct behavior. The development cycle is then based on the iteration of the following procedure:

- i. write a test known to fail according to the current stage of the implementation;
- ii. perform the smallest code update which satisfies all tests, including the one introduced in the previous point;
- iii. optionally refactor the produced code.

TDD makes testing the engine driving the overall development process: one of the hardest-to-find contributions for facilitators in an active programming learning context is suggesting a good *next test*. This has the role of letting pupils aware that their belief at a broad level (“the program works”) is false, thus an analogous belief at a smaller scale (for instance, “this function always returns the correct result”) should be false, too. This amounts to the destruction of knowledge necessary to build new knowledge (aka a working program) in a constructivist setting. Moreover, refactoring corresponds to the constructivist re-organization of knowledge following the discovery of more viable solutions: most of the developing activities consist in *realizing* that a system which was thought to correctly work is actually not able to cope with a new test case. This applies of course also to the simplest tasks faced by students engaged in learning the basics of computer programming.

Once pupils are convinced that their implementation is flawed, the localization of the code lines to be reconsidered is the other pillar of an active learning setting. Again, a paramount contribution for a successful learning process should be provided by a facilitator suggesting suitable debugging techniques (e.g., proposing critical input values, suggesting points in the execution flow to be verified, or giving advice about variables to be tracked during the next run).

Conclusions

The literature on learning to program through a constructionist strategy has often focused on how to bring the abstract and formal nature of programming languages into the manipulation of more concrete (or even tangible) “objects” (Kay *et al.*, 1997; Horn and Jacob, 2007; Dann, Cooper, and Pausch, 2008; Resnick *et al.*, 2009; Hauswirth, Adamoli, and Azadmanesh, 2017). Many proposals aim at overcoming the (initial) hurdles which textual rules of syntax may pose to children. Also, several environments have been designed in order to increase the appeal of programming by connecting this activity to real-world devices or providing fancy libraries. Instead, more work is probably needed to make educators and learners more aware of the so-called notional machine behind the programming language. Programming environments could be more explicit about the complex relationship between the code one writes and the actions that take place when the program is executed. Moreover, micro-patterns should be exploited in order to enhance problem solving skills of novice programmers, such that they become able to think about the solution of problems in the typical way that make the former suitable to automatic elaboration. Agile methodologies, now also common in professional settings, seem to fit well with constructionist learning. Besides the stress on teamworking, particularly useful seems the agile emphasis on having running artifacts through all the development cycle and the common practice of driving development with explicit or even executable “definitions of done”.

References

- Abelson, H., DiSessa, A.A. (1986). *Turtle Geometry: The Computer as a Medium for Exploring Mathematics*. Artificial Intelligence Series. AAAI Press.
- Abelson H., Dybvig R.K., Haynes C.T., Rozas G.J., Adams N.I., Friedman D.P., Kohlbecker, E. et al., (1998). Revised report on the algorithmic language scheme. *Higher-Order and Symbolic Computation*, 11(1), 7–105. <https://doi.org/10.1023/A:1010051815785>
- Abelson, H., Sussman, G.J., Sussman, J. (1996). *Structure and Interpretation of Computer Programs*. Second. MIT press.
- Allen, E., Cartwright, R., Stoler, R. (2002). DrJava: A lightweight pedagogic environment for Java. *SIGCSE Bull.*, 34(1), 137–41. <https://doi.org/10.1145/563517.563395>
- Astrachan, O., Wallingford, E. (1998). Loop patterns. In: *Proceedings of the Fifth Pattern Languages of Programs Conference*.
- Beck, K. (2003). *Test-Driven Development: By Example*. Addison-Wesley Professional.
- Beck, K., Andres, C. (2004). *Extreme Programming Explained: Embrace Change*. Second. Addison-Wesley Professional.
- Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J. et al. (2001). *Manifesto for Agile Software Development*. <http://agilemanifesto.org/iso/en/manifesto.html>
- Bell, T., Rosamond, F., Casey, N. (2012). Computer science unplugged and related projects in math and computer science popularization. In: Hans L. Bodlaender, Rod Downey, Fedor V. Fomin, and Daniel Marx (eds.), *The Multivariate Algorithmic Revolution and Beyond*. Berlin, Heidelberg: Springer-Verlag, 398–456. <http://dl.acm.org/citation.cfm?id=2344236.2344256>
- Bell, T., Vahrenhold, J. (2018). CS unplugged – how is it used, and does it work?. In: Böckenhauer, H.J., Komm, D., Unger, W. (Eds), *Adventures Between Lower Bounds and Higher Altitudes*. Lecture Notes in Computer Science, vol 11011. Springer, Cham.
- Bell, T., Lodi, M. (to appear). Constructing computational thinking without using computers. *Constructivist Foundations*.
- Belletтини, C., Lonati, V., Malchiodi, D., Monga, M., Morpurgo, A., Torelli, M. (2012). Exploring the processing of formatted texts by a kynesthetic approach. In: *Proc. of the 7th Wipscce*. WiPSCSE '12. New York, NY, USA: ACM, 143–44. <https://doi.org/http://dx.doi.org/10.1145/2481449.2481484>
- Belletтини, C., Lonati, V., Malchiodi, D., Monga, M., Morpurgo, A., Torelli, M. (2013). What you see is what you have in mind: constructing mental models for formatted text processing. In: *Proceedings of ISSEP2013*. *Commentarii Informaticae Didacticae 6*. Universitätsverlag Potsdam, 139–47. <http://opus.kobv.de/ubp/volltexte/2013/6368/pdf/cid06.pdf>
- Belletтини, C., Lonati, V., Malchiodi, D., Monga, M., Morpurgo, A., Torelli, M., Zecca, L. (2014). Extracurricular activities for improving the perception of informatics in secondary schools. In: Yasemin Gülbahar and Erin c Karatas (eds.), *Proceedings of ISSEP2014*. Lecture Notes in Computer Science, 8730. Springer, 161–72. https://doi.org/http://dx.doi.org/10.1007/978-3-319-09958-3_15
- Ben-Ari, M. (2001). Constructivism in computer science education. *Journal of Computers in Mathematics and Science Teaching*, 20(1), 45–73.
- Ben-Ari, M., Sajaniemi, J. (2004). Roles of variables as seen by Cs educators. *ACM Sigcse Bulletin*, 36(3), 52–56.
- Berg, J., (1999). Patterns for selection. In: *Proceedings of the 4th European Conference on Pattern Languages of Programs* (EuroPLoP '99). 305–326.
- Berry, M., Kölling, M. (2014). The state of play: A notional machine for learning programming. In: *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education*. ITiCSE '14. New York, NY, USA: ACM, 21–26. <https://doi.org/10.1145/2591708.2591721>
- Brin, D. (2016). *Why Johnny Can't Code*. https://www.salon.com/2006/09/14/basic_2/
- Clancy, M. (2004). Misconceptions and attitudes that interfere with learning to program. In: Sally Fincher and Marian Petre (eds.), *Computer Science Education Research*. Routledge, 85–100.
- Dann, W.P., Cooper, S., Pausch, R. (2008). *Learning to Program with Alice*. Prentice Hall Press.
- Dijkstra, E.W. (1985). On Anthropomorphism in Science. EWD936. <https://www.cs.utexas.edu/users/EWD/ewd09xx/EWD936.PDF>
- Du Boulay, B. (1986). Some difficulties of learning to program. *Journal of Educational Computing Research*, 2(1), 57–73. <https://doi.org/10.2190/3LFX-9RRF-67T8-UVK9>

- Feurzeig, W., Papert, S., Bloom, M., Grant, R., Solomon, C. (1970). Programming-languages as a conceptual framework for teaching mathematics. *SIGCUE Outlook*, 4(2), 13–17.
<http://dx.doi.org/10.1145/965754.965757>
- Findler, R.B., Bruce, R., Clements, J., Flanagan, C., Flatt, M., Krishnamurthi, S., Steckler, P., Felleisen, M. (2002). DrScheme: A programming environment for scheme. *Journal of Functional Programming*, 12(2), 159–82.
- Ford, J.L. (2009). Scratch programming for Teens. In: *Computer Science Books*.
- Goldberg, A., Kay, A. (1976). *Smalltalk-72 Instruction Manual*. Xerox.
- Goode, J., Chapman, G., Margolis, J. (2012). Beyond curriculum: the exploring computer science program. *Magazine ACM Inroads*.
- Hauswirth, M., Adamoli, A., Azadmanesh, M.R. (2017). The program is the system: introduction to programming without abstraction. In: *Proceedings of the 17th Koli Calling International Conference on Computing Education Research*. Koli Calling '17.
- Horn, M.S., Jacob, R.J.K. (2007). Designing tangible programming languages for classroom use. In: *Proceedings of the 1st International Conference on Tangible and Embedded Interaction*. TEI '07. New York, NY, USA: ACM, 159–62. <https://doi.org/10.1145/1226969.1227003>
- Kahn, K. (2017). A half-century perspective on Computational Thinking. In: *Technologias, Sociedade E Conhecimento*.
- Kastl, P., Kiesmüller, U., Romeike, R. (2016.) Starting out with projects: experiences with agile software development in high schools. In: *Proceedings of the 11th Workshop in Primary and Secondary Computing Education*. WiPSC '16. New York, NY, USA: ACM, 60–65. <https://doi.org/10.1145/2978249.2978257>
- Kay, A.C. (1993). The early history of smalltalk. *SIGPLAN Not*, 28(3), 69–95.
<https://doi.org/10.1145/155360.155364>
- Kay, A., Rose, K., Ingalls, D., Kaehele, T., Maloney, J., Wallace, S. (1997). Etoys & SimStories. *Walt Disney Imagineering*.
- Kurtz, T.E. (1978). BASIC. *SIGPLAN Not*, 13(8), 103–18. <https://doi.org/10.1145/960118.808376>
- Colleen, L., Esper, E., Bhattacharyya, V., Fa-Kaji, N., Dominguez, N., Schlesinger, A. (2014). Children's perceptions of what counts as a programming language. In: *Journal of Computing Sciences in Colleges*.
- Lonati, V., Monga, M., Morpurgo, A., Torelli, M. (2011). What's the fun in Informatics? Working to Capture children and teachers into the pleasure of computing. In: I. Kala and R.T. Mittermeir (eds.). In: *Proceedings of Iseep2011*, Lecture Notes in Computer Science, 7013. Springer-Verlag, 213–24. https://doi.org/10.1007/978-3-642-24722-4_19
- Maloney, J., Resnick, M., Rusk, N., Silverman, B., Eastmond, E. (2010). The Scratch programming language and environment. *Trans. Comput. Educ.*, 10(4), 16:1–16:15. <https://doi.org/10.1145/1868358.1868363>
- Meerbaum-Salant, O., Armoni, M., Ben-Ari, M. (2010). Learning computer science concepts with scratch. In: *Proceedings of the Sixth International Workshop on Computing Education Research*. 69–76.
- Missiroli, M., Russo, D., Ciancarini, P. (2016). Learning agile software development in high school: An investigation. In: *Proceedings of the 38th International Conference on Software Engineering Companion*, ICSE '16. New York, NY, USA: ACM, 293–302. <https://doi.org/10.1145/2889160.2889180>
- Piaget, J. (1973). *To Understand is to Invent: The Future of Education*. Penguin Books.
- Papert, S. (1980). *Mindstorms: Children, Computers, and Powerful Ideas*. New York, NY, USA: Basic Books, Inc.
- Proulx, V.K. (2000). Programming patterns and design patterns in the introductory computer science course. *ACM Sigcse Bulletin*, 32(1), 80–84.
- Qian, Y., Lehman, J. (2017). Students' misconceptions and other difficulties in introductory programming: a literature review. *ACM Trans. Comput. Educ.*, 18(1), 1:1–1:24. <https://doi.org/10.1145/3077618>
- Resnick, M. (1996). Distributed constructionism. In: *Proceedings of the 1996 International Conference on Learning Sciences*, ICLS '96. Evanston, Illinois: International Society of the Learning Sciences, 280–84. <http://dl.acm.org/citation.cfm?id=1161135.1161173>
- Resnick, M. (2007). All I Really Need to Know (About Creative Thinking) I Learned (by Studying How Children Learn) in Kindergarten. In: *Proceedings of the 6th Acm Sigchi Conference on Creativity & Cognition*, C&C '07. New York, NY, USA: ACM, 1–6. <https://doi.org/10.1145/1254960.1254961>
- Resnick, M. (2017). *Lifelong Kindergarten: Cultivating Creativity Through Projects, Passion, Peers, and Play*. MIT Press.
- Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A. et al. (2009). Scratch: programming for all. *Commun. ACM*, 52(11), 60–67.
<https://doi.org/10.1145/1592761.1592779>

- Rodger, S., Hayes, J., Lezin, G, Qin, H., Nelson, D., Tucker, R., Lopez, M., Cooper, S., Dann, W., Slater, D. (2009). Engaging middle school teachers and students with alice in a diverse set of subjects. *SIGCSE Bull.*, 41(1), 271–275. <https://doi.org/10.1145/1539024.1508967>
- Sajaniemi, J. (2002). An empirical analysis of roles of variables in novice-level procedural programs. In: *Human Centric Computing Languages and Environments, 2002. Proceedings. IEEE 2002 Symposia on*. IEEE, 37–39.
- Sirkkiä, T. (2012). *Recognizing Programming Misconceptions: An Analysis of the Data Collected from the Uuhiste Program Simulation Tool*. Master’s thesis, Department of Computer Science; Engineering, Aalto University.
- Slany, W. (2014). Tinkering with Pocket Code, a Scratch-like programming app for your smartphone. In: *Proceedings of Constructionism 2014*.
- Sorva, J. (2013). Notional machines and introductory programming education. *Trans. Comput. Educ.*, 13(2), 8:1–8:31. <https://doi.org/10.1145/2483710.2483713>
- Taub, R., Armoni, M., Ben-Ari, M. (2012). CS unplugged and middle-school students’ views, attitudes, and intentions regarding CS. *TOCE*, 12(2), 8. <https://doi.org/10.1145/2160547.2160551>
- Tumlin, N. (2017). Teacher configurable coding challenges for block languages. In: *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*.
- Vygotsky, L. (1978). *Mind in Society*. London: Harvard University Press.
- Weintrop, D., Uri, W. (2015). To block or not to block, that is the question: students’ perceptions of blocks-based programming. In: *IDC '15 Proceedings of the 14th International Conference on Interaction Design and Children*.
- Wirth, N. (1993). Recollections about the development of Pascal. *SIGPLAN Not.*, 28(3), 333–42. <https://doi.org/10.1145/155360.155378>



M. Lodi is a PhD student in Computer Science, Department of Computer Science and Engineering, University of Bologna, Italy. He also received Bs, Ms and High school teaching licence in CS from the same University. He works on computer science education, with a particular focus on teacher training about computational thinking and epistemological aspects of Computer Science as a discipline. In particular, he studies “Computer Science Growth Mindset”. He published some papers in international conferences on computer science education, and a book in Italian for primary school teachers. He is actively involved in nation-wide initiatives to introduce CS in Italian K-12 curriculum. <https://lodi.ml>



D. Malchiodi (<http://malchiodi.di.unimi.it>) is an Associate Professor at Università degli Studi di Milano (Department of Computer Science), where he teaches “Statistics and data analysis” and “Big scale analytics”. His research activities are focused on the one hand on the treatment of uncertainty in machine learning problems, and on the other one on the development of teaching methodologies for primary and secondary education. He published around one hundred papers and he participated in the activities of ten research projects and research groups, at national and international level. He is co-founder of the ALaDDIn working group (<http://aladdin.unimi.it>), involved in several activities focused on the popularization of computer science, including the training of secondary school teachers and a radio broadcast on informatics culture.



M. Monga is an Associate Professor at Università degli Studi di Milano (Department of Computer Science). His research interests are mainly in the field of software engineering, system security, and computer science education. Since he believes it is urgent to change the common misconception of informatics as the mere use of information technologies, he founded together with Carlo Bellettini, Violetta Lonati, Dario Malchiodi, and Anna Morpurgo a group working to spread informatics as a science among the general public (<https://aladdin.unimi.it/>). Moreover, he is the National Bebras Organizer for Italy.



A. Morpurgo is Assistant Professor at the CS Department, Università degli Studi di Milano, Italy. Her current research interests are mainly in CS education. She is actively involved in nation-wide initiatives to introduce CS in the Italian K-12 curriculum and is co-founder of the ALaDDIn group (<http://aladdin.unimi.it>), working on the popularization of informatics as a science in school and among the public. She is involved in the training of secondary school teachers and is part of the team organizing the Bebras challenge in Italy.



B. Spieler has a PhD in Engineering Sciences. She is a University Assistant at Graz University of Technology, Department for Software Technology. Her work is focused on how to encourage female teenagers with playful coding activities with the Pocket Code app or in “Girls Coding Weeks”. Moreover, her recent work is related to gender, game based/mobile learning and constructionist gaming. Through the nonprofit university project Catrobat (<https://catrobat.org>), Mrs. Spieler promotes computational thinking skills in a fun and engaging way among children, teenagers and teachers on a worldwide scale. <https://bernadette-spieler.com>

Survey and Analysis of Computing Education at Japanese Universities: Subject of “Information” for High School Teacher’s License*

Kazuhiro SUMI¹, Mika OHTSUKI², Tetsuro KAKESHITA²

¹*Faculty of Education, Saga University, 840-8502, Japan*

²*Faculty of Science and Engineering, Saga University, 840-8502, Saga, Japan*

e-mail: sumik@cc.saga-u.ac.jp, mika@is.saga-u.ac.jp, kake@is.saga-u.ac.jp

Abstract. We conducted the first national survey of computing education at Japanese universities in 2016. In this paper, we report the survey result of the computing education to obtain high school teacher’s license on IT. The survey covers various aspects of computing education including program organization, quality and quantity of educational achievement, students, teaching staff and computing environment. We collected 338 answers through the survey which cover 65% of the departments having teacher’s license course on IT. Many of the responded departments also provide computing education majored in computing discipline. Although 5,006 students are enrolled in the computing education for the license, only 369 students obtain the license since very few are employed at a high school. Most of the teacher’s license holders on computing subject also obtain high school teacher’s license of other subject in order to get a job as a high school teacher.

Keywords: computing education, subject of information, high school, teacher’s license.

1. Introduction

Computing education is essential at modern universities. There are four types of computing education in Japanese universities:

- A. Computing education at a department or a course majored in computing discipline.
- B. Computing education at a non-IT department or a course as a part of their major field of study.

* This paper is a revised and extended version of the following paper written by the same author K. Sumi and T. Kakeshita, “National survey of Japanese universities on computing education: Analysis of IT education for high school teacher’s license on IT”, in Proc. 12-th International Conference on Digital Information Management (ICDIM 2017), pp. 87–92, 2017.

- C. General computing education for all students at a university or a faculty typically at the first or second academic year.
- D. Computing education to obtain high school teacher's license on computing subjects.

The Science Council of Japan announced the reference standard of informatics (Hagiya, 2015) for university education in March 2016. The reference standard provides a common body of knowledge (BOK) for college level computing education and the Japanese government accepted this as the definition of computing education.

This survey is designed to analyze and understand current status of computing education at Japanese universities from various aspects including program organization, quality and quantity of educational achievement, students, teaching staff and computing environment.

In this paper, we report and discuss the result of the survey type D for computing education to obtain high school teacher's license on computing subject. The Enforcement Regulations of the Japanese Education Official License Law defines requirements for an education program to issue a regular high school teacher's license for the subject "Information" in chapter one, named Method of Learning Units, article 5. It is necessary for a program to include the following six subjects:

1. Information society and information ethics.
2. Computer and information processing including practical training.
3. Information system including practical training.
4. Information communication network including practical training.
5. Multimedia expression and technology including practical training.
6. Information and occupation.

A program needs to be accredited by the Japanese government to fulfill the requirements. The Japanese ministry of education (MEXT) maintains the list of the accredited programs.

We have already published the survey outline in (Kakeshita, 2017b) . The results of other survey types are also published as separate papers (Kakeshita, 2018; Kakeshita and Kakeshita, 2017; Kakeshita, 2017a; Ohtsuki *et al.*, 2017). Information processing society of Japan (IPSJ) will utilize the survey result to develop the new J17 curriculum standard (IPSJ, 2018). MEXT will utilize the survey result to improve the national policy of computing education.

2. Survey Outline

2.1. Survey Questions

The following is the list of questions for survey type D. As the reader can understand from the list, our survey covers various aspects of computing education by considering the Japanese standard for establishment of universities and our experience of accrediting computing programs in Japan.

- Name of university, faculty, department and course.
- Program organization:
 - Day time, night or remote program.
 - Academic discipline of the program such as engineering, social science and humanities.
 - Required number of credits of computing subjects for graduation.
 - Number of computing subjects provided.
 - Classification of the computing subjects.
- Quality and quantity of educational achievement:
 - See Section 2.2 for detail.
- Student:
 - Regular academic years of computing education.
 - Quota, Number of enrolled students.
 - Number of students who obtain the license.
 - Student's choice of career after graduation.
- Teaching staff:
 - Number, educational background, current specialized field, tenure of faculty members.
 - Number and workload of support staff and teaching assistant students.
- Computing environment:
 - Educational computer system.
 - Student's own PC and utilization at class.
 - Educational programming language.
- Other topics:
 - Future plan and strength of the program.
 - Utilization of IT certification and/or qualification.
 - Special remarks.

2.2. Survey of Quality and Quantity of Educational Achievements

The survey of quality and quantity of educational achievements is the core of our survey. We define six achievement levels for knowledge and skill represented in Table 1. These levels are used to describe educational achievement.

We also define a BOK based on the reference standard of informatics (Hagiya, 2015) and additional topics related to general computing education (Kawamura, 2008). The BOK contains 90 topics classified by 21 domains as represented in Table 2. The BOK is used to precisely define educational achievement of each program. The numbers within the parenthesis are the number of topics belonging to the section or the domain.

We adopted the same definition of level and BOK throughout the survey types A to D in order to enable mutual comparison of the different survey types. Such comparison is important to understand relationship among different survey types.

In case of survey type D, a department or a course responds to the survey.

Table 1
Knowledge and Skill Level Definition

Level	Knowledge Level Definition	Skill Level Definition
0	Not taught (unnecessary or already taught at general computing education)	
1	Not taught because of the time limitation or because the level of the contents is too high	Taught at class with simple exercise
2	Taught at class. Students know each term	Taught at class with some exercise. Students can perform the topic if detailed instruction is provided
3	Taught at class. Students can explain the meaning of each term	Taught at experiment with more complex exercise. Students can perform the topic with simplified instruction
4	Taught at class. Students can explain relationship and/or difference among related terms	Students perform combined research project containing the topic so that the students can autonomously perform the topic
5	Taught at class or graduation research project. Students can teach related domain or subject of the terms to the others	Students perform combined research theme containing the topic. Students can teach how to perform the topic to others

Table 2
Common BOK Organization

Source	Section	Domain
J07-GEBOK	General Education	Informatics in General Education (9)
Reference Standard of Informatics	(A) General Principles of Information	(6)
	(B) Principles of Information Processing by Computers	Information Transformation and Transmission (4), Information Representation, Accumulation and Management (4), Information Recognition and Analysis (4), Computation (6), Algorithms (8)
	(C) Technologies for Constructing Computers that Process Information	Computer Hardware (3), I/O Device (4), Fundamental Software (3)
	(D) Understanding Humans and Societies that Process Information	Process and Mechanism for Information Creation and Transmission (2), Human Characteristics and Social System (3), Economic System and Information (2), IT-based Culture (2), Transition from Modern Society to Post Modern Society (2)
	(E) Technologies and Organizations for Constructing and Operating "Systems" that Process Information in Societies	Technics for Information System Development (7), Technics to Obtain Information System Effect (6), Social System Related to Information (2), Principle and Design Methodology for HCI (4)
	Competence	Professional Competency for IT Students (3), Generic Skill for IT Students (6)

2.3. Survey Process

We prepared the survey in October 2016. We defined the survey questions and set up the web-based survey system (Kakeshita and Ohtsuki, 2011). We utilized the web-based survey system since we did not exactly know the actual organization for this survey in advance. After preparing various document such as user manual and detailed explanation of the survey questions, we sent the formal request letter to all universities in Japan with a reference letter from the Japanese Ministry of Education in order to increase the response rate.

The survey was executed for two months starting at the beginning of November 2016. Each survey responder must first register to the web-based system and then answer the questions listed in Section 2.1. We also provide FAQ and independent answers for the questions from the responders.

After closing the survey, we reviewed the collected answers and requested the responders for possible correction of the incomplete answers.

3. Response Rate Analysis of the Survey

The target of survey type D is a department or a course which provides an educational program to obtain high school teacher's license on IT. Such educational program must be accredited by the Japanese ministry of education. We collected data related to computing subjects and the subjects to teach handling of computer. If there are more than one courses of the teacher training of high school subject "information" on the same campus, each of the course is requested to respond to the survey after the independent registration to the survey web site.

We designated the names of universities, undergraduates, departments (or courses) corresponding to the teaching professionals for the answer of target organizations. The resulting number of responses is shown in Table 3. Here a public university is founded and supported by a local government such as prefecture or big city. We also counted the number of courses from the responses since some universities merged multiple courses within a response. The number of accredited courses is counted from the list of the accredited courses published by the Japanese ministry of education. The surveyed year is the 2016 fiscal year.

Table 3
Number of Responses to Survey Type D

University Type	Number of Responses	Number of Courses	Number of Accredited Courses
National	85	75	107
Public	18	14	17
Private	235	251	397
Total	338	340	521

4. Program Organization

There are three types of education programs for high school teacher's license on subject "Information". The first one is developed within a computing department. The second one is developed within a non-computing department having a computing department within the same university. The third type is developed at a university without computing department. Table 4 represents the number of program for the three cases. We shall analyze the survey result based on this classification.

We collected answers on corresponding subjects for the teaching curriculum of high school subject "information" or the operations for information equipment. The results are shown in Table 5. Most of the subjects are taught as a lecture.

Table 5 shows subjects corresponding to 20 credits of "Subjects related to computing domain" defined in the license law enforcement regulations for the accreditation criteria of the Japanese Ministry of Education.

The remarks on subjects related to computing domain are as follows:

1. Credits earned for more than 20 credits in "Subjects related to computing domain" are included in "Subjects related to computing domain or teaching activity".
2. If you acquire credits in "Information Processing" and "Computer Network Theory" in "Subjects related to computing domain", the required credits for "Subjects related to computing domain or teaching activity" will be 10 or 8 credits depending on the earned credits.
3. "Subjects based on Article 66-6 of the License Law Enforcement Regulations" and "Subjects related to computing domain" can be used for graduation.
4. Credits for "Subjects related to computing domain" can also be earned according to the rule for daytime course.

Fig. 1 represents distribution of the number of experiments in the accredited programs. As shown in the Fig. 1, 32 programs (20.2%) in the computing departments have more than two experimental subjects. However, the number of programs providing experiments decreases at the programs in non-computing department. Particularly there is only one program (1.3%) providing experiments at a university having no computing department. There is a significant difference between computing department and non-computing department will be discussed in the succeeding sections.

Table 4
Classification of Programs based on Supporting Department

Classification	# of Responses
Developed in a Computing Department	158
Developed in a non-Computing Department having Computing Department within the same University	106
No Computing Department with the University	74

Table 5
Subjects related to Computing Domain in the License Law

Subject Category Defined in the License Law	Number of Credits	Subject Name	Required Credit	Elective Credit
Information Society and Information Ethics	20	Social information theory	2	
		Organizational Information Theory		2
Computer and Information Processing (including practice)		Information Processing Basics	2	
		Information Processing	2	
		Information Mathematics		2
		Software Science		
		Planning Science	4	
Information System (including practice)		Decision-Making Theory		4
		Information System Theory	2	
		Information System Construction Theory		2
		Information System Management Theory		2
Information System Management Theory		Management System Basics		2
		Computer Network Theory	2	
Information Communication Network (including practical training)				
Multimedia Expression and Technology (including practice)		Digital Design Theory	2	
		Operations Research	2	
Information and Occupation		Information and Occupation	2	
		Business System Theory		2
Required Credits	20	□	20	

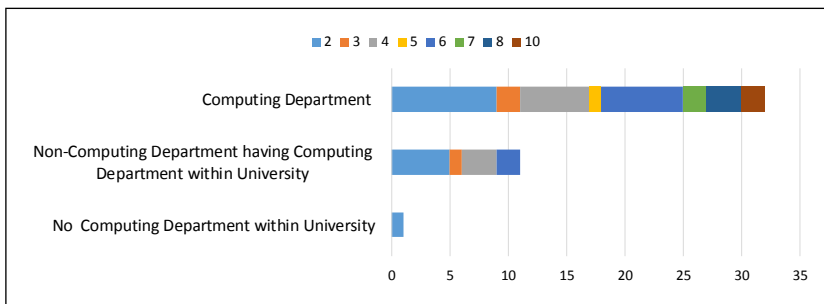


Fig. 1. Number of Experiments included in the Accredited Programs.

5. Quality and Quantity of Educational Achievement

In this section, we outline the educational achievement of the program and the survey results of the education level.

For each of the sections defined in Table 2, we define effort by the sum of the multiplication of the number of enrolled students and the level value of each domain included

in the answers. These allow estimating the effort that each institution is spending for a combination of each domain and knowledge/skill.

We first remove outliers of the collected data using the IQR method. IQR (interquartile range) is defined by the difference between the first and the third quartile of the collected data.

$$\text{IQR} = \text{Data Value at the 3-rd Quartile} - \text{Data Value at the 1-st Quartile}$$

Next, we calculate the value obtained by adding 1.5 times of the IQR to the third quartile. Data above this value are considered outliers. Also, we calculate the value obtained by subtracting 1.5 times of the IQR from the first quartile. Data below this value are also considered outliers. If the data is completely normal distributed, then IQR is standard deviation (SD) multiplied by 1.35. The third quartile is the average SD multiplied by 0.67, so an average SD multiplied by 2.70 plus 1.5 times IQR is the top outlier division. The summary result is shown in Fig. 2. This shows an overview of the areas which educational institutions are focused on.

The knowledge effort and skill effort have some differences, but we find a very high correlation coefficient value of 0.97. The results of sorting the areas in descending order of knowledge effort are shown in Table 7. This is considered to represent the importance of each domain recognized by the educational institution. In addition to this, the effort value of “general IT education” and “generic skills for IT students” are high, but this is due to the lack of teachers who can handle full-fledged information specialized education. The relative decrease in average academic achievement of university students is estimated as the rate of increase in the background.

While the effort ratio to teach general education is high, but the average achievement level is not high which compared with other regions. This reason is that the general IT education is often taught in the first or second academic year for all college students.

The effort ratio is greater for “generic skill for IT students” is large, and the average of the achievement levels for this skill is higher compared to other regions. This reason

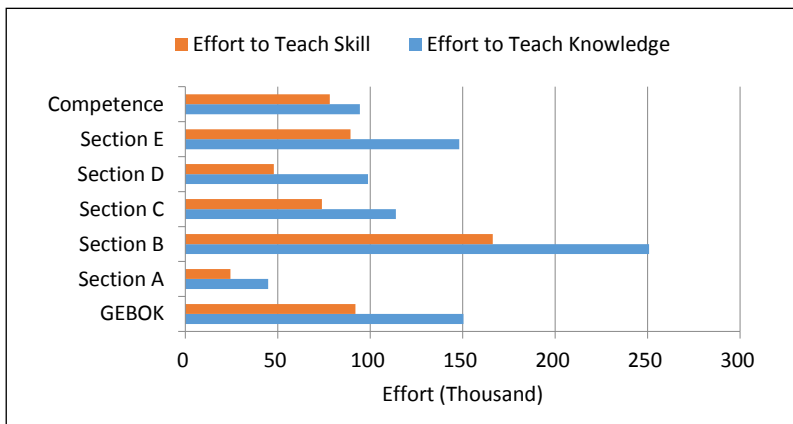


Fig. 2. Effort distribution to Teach Knowledge and Skill.

Table 7
Effort to Teach Knowledge and Skill of Each Domain

Domain Name	Effort to Teach Knowledge	Effort to Teach Skill
General IT Education	150,361	92,050
General Principles of Information	44,729	24,311
Information Transformation and Transmission	43,112	26,518
Information Representation, Accumulation and Management	56,562	38,215
Information Recognition and Analysis	33,918	23,651
Computation	49,004	31,440
Algorithms	68,280	46,503
Computer Hardware	30,166	20,079
I/O Devices	44,562	28,090
Fundamental Software	39,087	25,731
Process and Mechanism for Information Creation and Transmission	21,695	10,849
Human Characteristics and Social System	24,231	11,234
Economic System and Information	14,164	6,184
IT-based Culture	21,658	11,776
Transition from Modern Society to Post Modern Society	17,103	7,801
Technics for Information System Development	54,459	39,503
Technics to Obtain Information System Effect	36,878	18,900
Social System Related to Information	29,805	13,752
Principle and Design Methodology for HCI	27,009	17,152
Professional Competency for IT Students	33,488	23,749
Generic Skill for IT Students	60,963	54,332
Information processing, calculation, data analysis	389	64

is that the generic skills are often educated in college subjects including graduation research.

Next, we shall analyze the quality of education. Figures Fig. 3-1 through Fig. 9-2 show the distribution of the number of students at each knowledge and skill levels for each section defined in Table 2. Since the total number of students is different, we shall show the ratio of the number of students. The distributions are shown in three cases defined in Table 4. The purpose of the comparison is to clarify the impact of the computing department, since computing departments are expected to have more teaching resource, such as teaching staff and computing facility, than other departments.

Fig. 3-1 and Fig. 3-2 represent the distributions for General IT Education. The knowledge levels in Fig. 3-1 are concentrated between two and four, especially in the computing department, where the level four is the highest. In the skill level shown in Fig. 3-2, nearly 40% of the Case 2 show level 0, while Level 2 is the highest for Case 3. This also demonstrates advantage of computing department even for general IT education.

Fig. 4-1 and Fig. 4-2 represent the distribution of student numbers against knowledge and skill levels achieved in the three cases of the section A of the reference standard. Readers can find that the mean values are similar in the three cases, but Case 1 has a larger standard deviation than Case 2 or Case 3. Case 1 is also the highest at level 4 in Fig. 4-1 and Fig. 4-2.

- **Case 1:** Computing Department
- **Case 2:** Non-Computing Department having Computing Department within University
- **Case 3:** No Computing Department within University

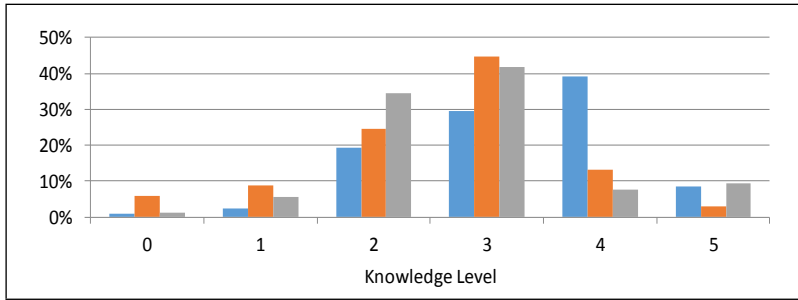


Fig. 3-1. Knowledge Level Distribution (General IT Education).

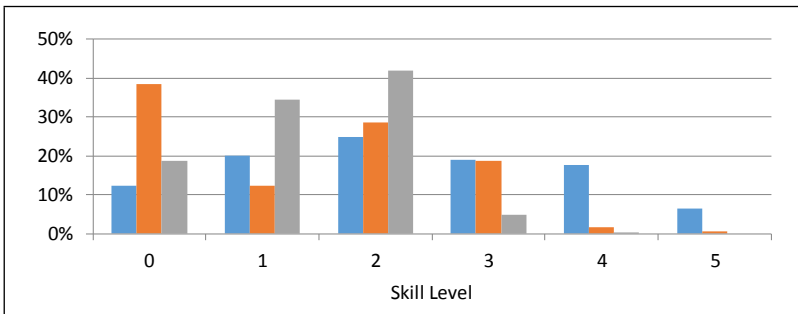


Fig. 3-2. Skill Level Distribution (General IT Education).

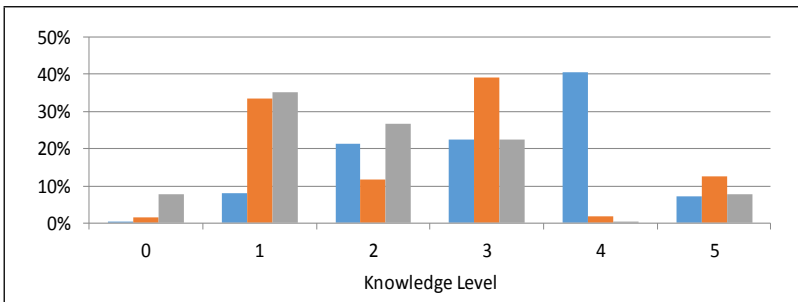


Fig. 4-1. Knowledge Level Distribution (Section A of the Reference Standard).

Fig. 5-1 and Fig. 5-2 show a comparison of knowledge and skill level distributions on the principles of information procession by computers (Section B of the Reference Standard) for the three cases. The distribution of knowledge levels in Fig. 5-1 shows a

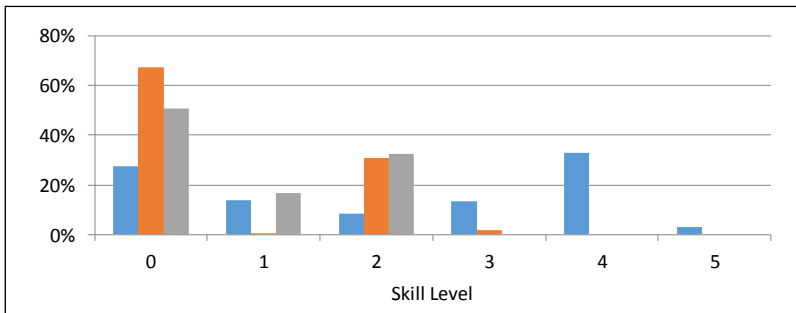


Fig. 4-2. Skill Level Distribution (Section A of the Reference Standard).

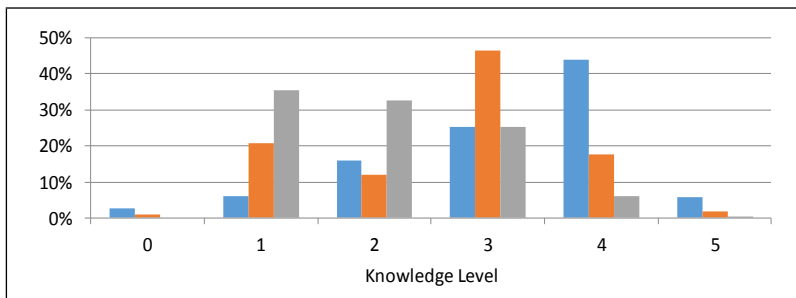


Fig. 5-1. Knowledge Level Distribution (Section B of the Reference Standard).

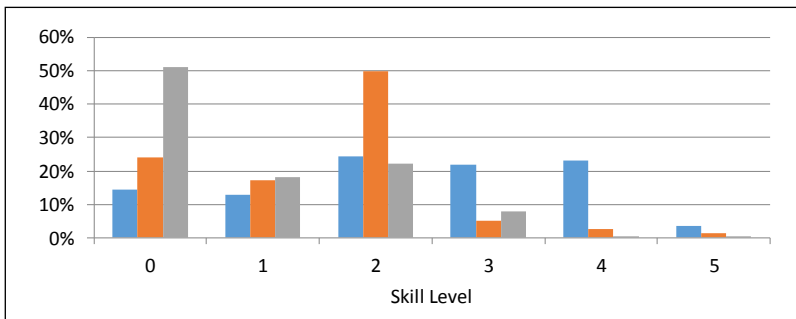


Fig. 5-2. Skill Level Distribution (Section B of the Reference Standard).

similar trend, but 40% of Case 1 supports Level 4. There is a big difference in the distribution of skills in Fig. 5-2. In Case 1, 50% of the students have achieved a skill level greater than 3, but in Case 2 and Case 3, 50% or more of the students have skill level 0. This is considered an impact of the computing department.

Fig. 6-1 and Fig. 6-2 show a comparison of knowledge and skill level distributions on the technologies for constructing computers that process information (Section C of the Reference Standard). At the knowledge level shown in Fig. 6-1, more than 50% of Case 2 and 3 are level 2, while the peak of Case 1 is level 4. The distribution of skill lev-

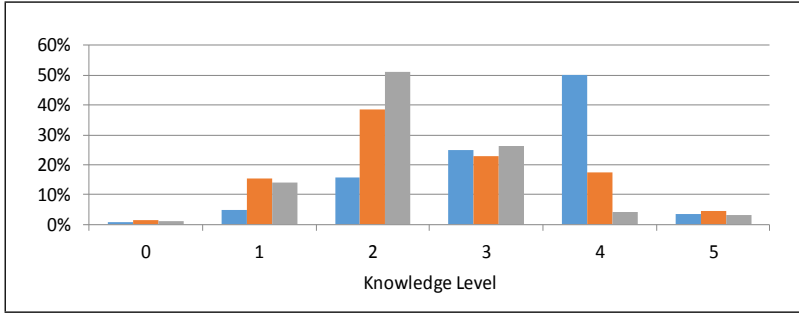


Fig. 6-1. Knowledge Level Distribution (Section C of the Reference Standard).

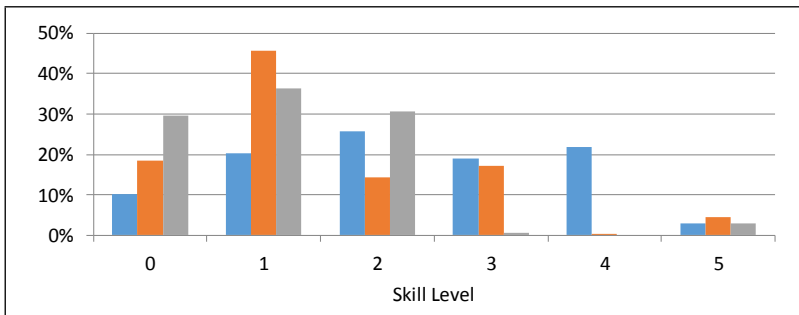


Fig. 6-2. Skill Level Distribution (Section C of the Reference Standard).

els in Fig. 6-2 shows a similar trend. Achievement of both Knowledge and Skill Levels are higher in Case 1 than the other two cases.

Fig. 7-1 and Fig. 7-2 show a comparison of the distribution of knowledge and skill levels about understanding humans and societies that process information in the three cases (Section D of the reference standard). In the knowledge level shown in Fig. 7-1, 35% of Case 1 shows Level 4, and the skill level shown in Fig. 7-2 shows 30% of Case 1 is at level 4. In both skill and knowledge, the readers can observe that Case 1 has a higher level.

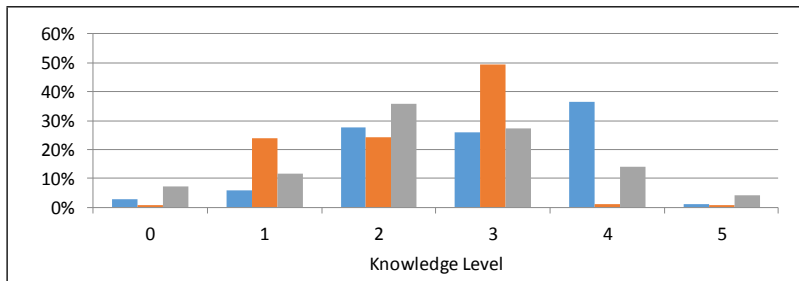


Fig. 7-1. Knowledge Level Distribution (Section D of the Reference Standard).

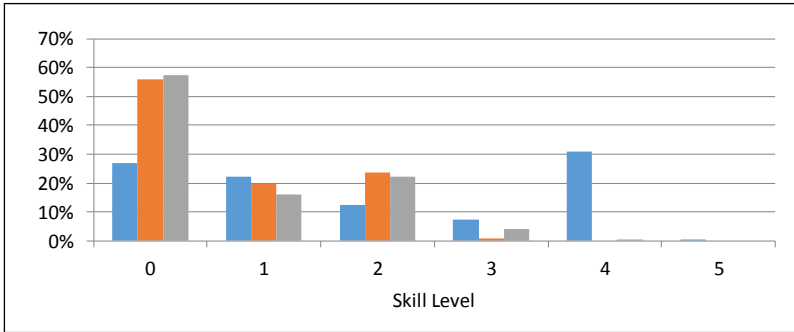


Fig. 7-2. Skill Level Distribution (Section D of the Reference Standard).

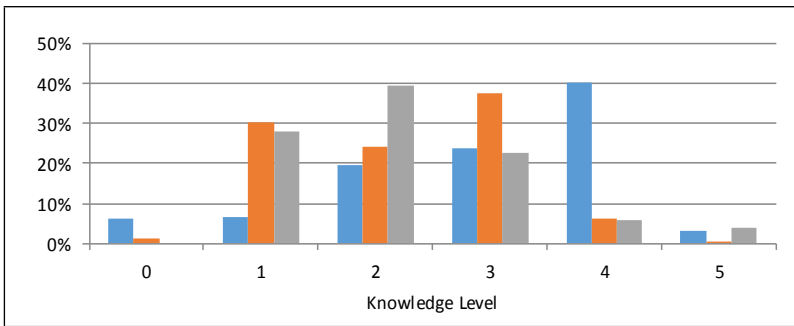


Fig. 8-1. Knowledge Level Distribution (Section E of the Reference Standard).

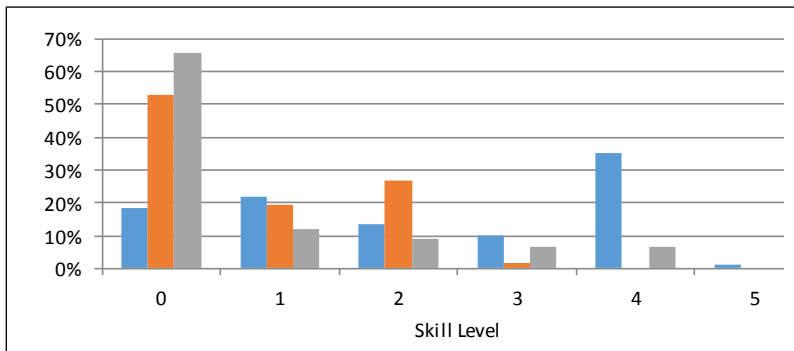


Fig. 8-2. Skill Level Distribution (Section E of the Reference Standard).

Fig. 8-1 and Fig. 8-2 show a comparison of the distribution of knowledge and skill levels for technologies and organizations for constructing and operating “systems” that process information in societies (Section E of the reference standard). At the knowledge level in Fig. 8-1, 40% of Case 1 indicates Level 4, and the skill level in Fig. 8-2 also indicates 35% of Case 1 at Level 4. Also, in Cases 2 and 3, we found that more than 50% of the students are not taught anything about skills.

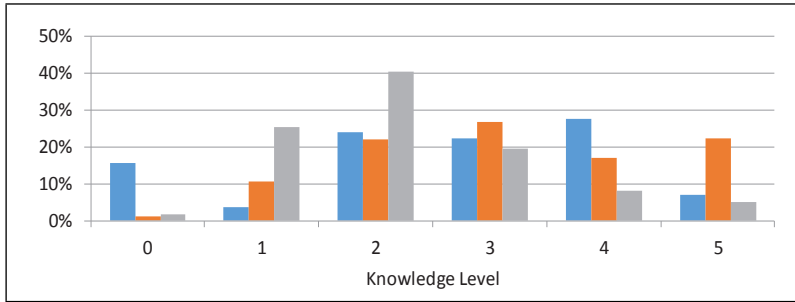


Fig. 9-1. Knowledge Level Distribution (Competence of the Reference Standard).

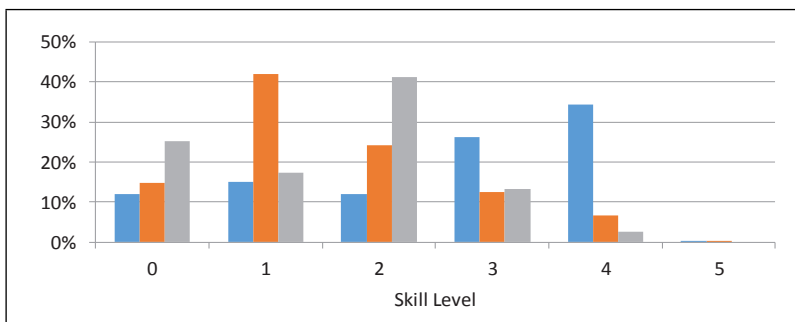


Fig. 9-2. Skill Level Distribution (Competence of the Reference Standard).

Fig. 9-1 and Fig. 9-2 show a comparison of the knowledge and skill level distributions of competence section of the reference standard of informatics. In Fig. 9-1, 28% of Case 1 has reached to level 4, whereas Case 3 has a lower peak of 39% at level 2. In Fig. 9-2, while 35% of Case 1 is at level 4, the peaks in Case 2 and Case 3 are level 1 and 3 respectively, which are significantly lower than Case 1.

As the reader can understand from Fig. 3-1 to Fig. 9-2, the educational achievement of Case 1 is generally higher than Case 2 and Case 3. This is considered an effect of the computing department, as the computing department usually hires more computing professionals as faculty members. This shows the importance of teachers in charge of computing education.

6. Students

Fig. 10 shows the distribution of the standard academic year of the subject “information” for teacher training. In most cases, the teacher training course is provided for 1–3 or 1–4 academic years.

In the distribution of student quota at each educational program, **the sum of the student quota is 20,854**. It indicates the maximum number of students who can obtain the

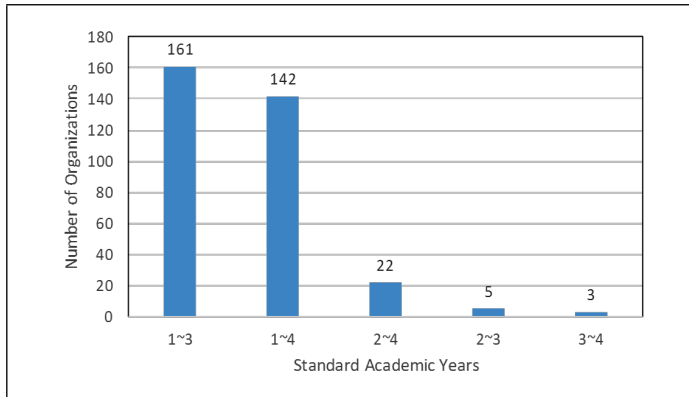


Fig. 10. Standard Academic Years N = 338.

teacher’s license at each program. We also collected the number of enrolled students at each program. The number of enrolled students represents the average of the last three years. The total number of students who obtained the license is 369 (275 males and 94 females). This means that most of the program issue very small number of teacher’s license compared to their quota.

Fig. 11 shows the number of enrolled students at each program. The number of enrolled students at each program never exceeds 10 so that all education programs are very small. There are many programs with no enrolled students. Some of the programs quitted to issue teacher’s license. Total student enrollment is 5,011 (863 males and 4,143 females). The readers can observe that the number of students obtained teacher’s license is quite few compared with the number of student enrollment. This

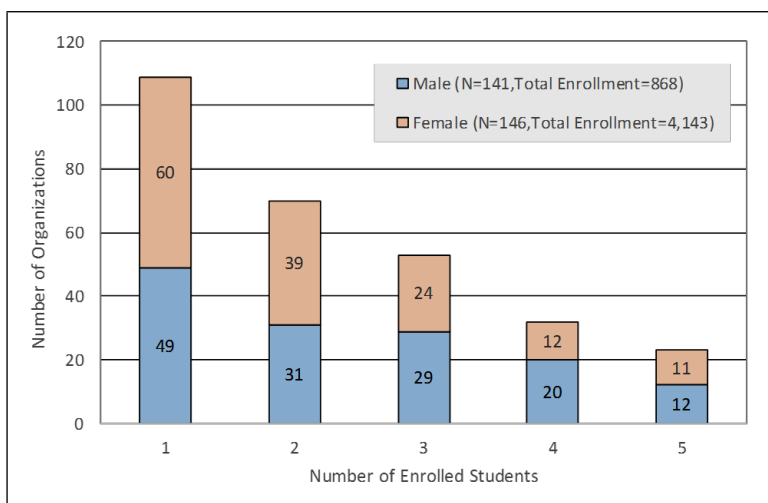


Fig. 11. Student Enrollment.

is mainly because that the number of open positions of IT teachers at high school is quite limited.

Table 8 represents students' career selection after graduation. Students who took a job as a teacher of the subject "information" including a temporary adoption remained a total of 198 persons (1.1%). On the other hand, 1,290 students (7.4%) took a job as a teacher other than the subject "information". In addition, 3,173 students (18.3%) enter graduate schools, while 11,371 students (65.5%) are hired at a company or a government.

Although the number of teachers with high IT skill and experience is limited at high school, the new open position for the high school teacher of subject "Information" is very small as explained above. The reason is that the number of teachers to teach "Information" is typically only one at each high school and there are many cases that a high school teacher with license of other subjects such as mathematics is already teaching subject "Information" at high school. Thus, a student often obtains double license such as subject "Information" and "Mathematics" in order to increase possibility to get job at high school. It is expected to improve such situation since computing subject is one of the core knowledge and skill at the 21-st century.

Table 8
Student's Career Selection after Graduation

Career Selection	Number of Students	Ratio (%)
High School Teacher of Subject "Information"	198	1.1
High School Teacher of Another Subject	1,290	7.4
Graduate School of Computing Discipline	1,803	10.4
Graduate School of other Discipline	1,370	7.9
Hired at Company or Government	11,371	65.5
Others (including unknown)	1,332	7.7
Total	17,364	100

Table 9
The total number of persons and representative class

Type of Faculty Member	Total Number of Persons	Number of Faculty Graduated IT Department	Number of Faculty Members Majored in Informatics	Total Number of Classes in Charge
Full-Time Teacher without a Term of Office	319	214	241	250
Full-Time Teacher with Term of Office	104	52	74	95
Admiral · Cumber some Staff (in-House Teacher)	130	74	86	120
Part Time Lecturer (Outside School)	231	140	166	218

7. Teaching Staff

Table 9 represents the total number of persons and the number of representative class of person in charge, assistant of the subject “information” teacher training. We find teachers who did not graduate a department majored in computing discipline or whose current major is not informatics. Faculty development for these teachers is quite important.

8. Computing Environment

Table 10 represents the answers of the educational computer system utilized by the educational program. It should be noted that 106 programs (31.4%) do not have educational computer system. Since we have found that the non-existence of the educational computer system greatly affects students’ skill level (Kakeshita, 2018), improvement is strongly recommended.

Table 11 represents the utilization of student PC at the programs. Most of the courses do not require or recommend their students to purchase or possess PC for use in the classroom.

Table 10
Utilization of Educational Computer System

Selection	Number of responses
Shared Use of Educational Computer System at University	113
Shared Use of Campus Educational Computer System	39
Shared Use of Educational Computer System at Faculty	22
Using the Department’s Educational Computer System	47
There is an Educational Computer System in the University, but They are not Used for the Education	11
There is no Educational Computer System in the University	106

Table 11
Utilization of Student PC

Utilization	Number of Responses
All students of the university must have PC	25
All students of the faculty must have PC	24
All students of the department/course must have PC	20
Students are recommended to purchase PC	25
Purchasing of Student’s own PC is optional	244

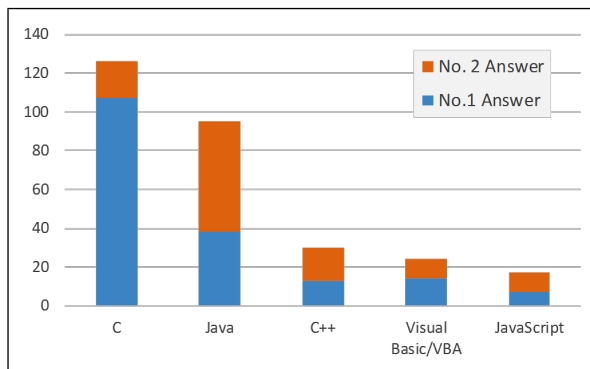


Fig. 12. Popular Educational Programming Language.

11 programs do not use existing educational computer system. Most of these programs require students to purchase their own PC. There is no educational computer system at 106 universities. Further investigation of the computing education is required for the latter case.

We asked to answer educational programming languages which the courses are using. The programming languages are selected such that the student reached a level beyond the level to understand a simple program written in that language. Fig. 12 shows the top-5 languages with the highest and the second highest student achievement level at each program.

9. Other Effort Related to Computing Education at Individual Programs

9.1. Future Plans

We collected the answers from different departments about their future plan during the survey. We shall introduce some of them.

Some departments are preparing to set up subjects such as “digital marketing” and “digital business modeling” related to “business” and “information”. Some reorganize the Information Engineering Department and the Electrical and Electronic Engineering Department into the “Electronic Information Systems Engineering Department”, revise the curriculum considering recent advancement of AI, IoT, Big Data, etc. Some establish Big Data Course. Many departments have a plan of curriculum revision to accommodate recent technology change.

As a concrete example, there is a case that reorganized in Science and Engineering Faculty to create intelligent information system course in 2017, so that the students can learn the state-of-the-art technology such as artificial intelligence that can learn various calculations such as numerical analysis and optimization from basic principles. There was also a department that nurtured human resources that would be the driving force

that drives the center of society through education aimed at incorporating it. In addition, there is a department to consider whether to use C or Java as the main language, and to consider not only PHP but also Ruby and Python as CGI.

9.2. Distinctive Practices

Common features of each educational program include accreditation from JABEE (Japan Accreditation Board for Engineering Education) in many departments, designing curriculum based on ACM/IEEE Computing Curriculum Guidelines and J07 Computing Curriculum Standard developed by IPSJ (Information Processing Society of Japan). There are also departments that discuss high school-high school collaboration and some departments provide educational materials and IT materials for elementary and junior high schools.

In addition, some introduce a program of informatics into general education, arrange programming languages to be able to learn in cooperation from 1 to 3 academic years. There are some departments providing simultaneous teaching license of "informatics" and "mathematics". There was something we could do to improve the educational program.

Furthermore, some departments are promoting computing education by introducing e-learning by Moodle etc. There were cases where qualified instructors were enrolled and promoting the acquisition of "IT passport" or "P inspection grade 2".

As a concrete example, some departments set importance on related subjects such as big data, data mining, security, and established the next-generation robot laboratory, which enriched the subjects of mathematics, especially statistics, as the foundation, and collaborated with companies. A department actively promotes research and acquisition of external funds. The Ministry of Education, Culture, Sports, Science and Technology (MEXT) has been adopted as a collaborating university in the field of security in the "Formulation of Information Technology Human Resources Development Center Supporting Growth Areas (enPiT)". Based on this project, some departments have newly established one PBL-type subject for third-year undergraduate students from 2017 and reinforced practical education on big data processing, AI, and cloud technology.

In order to learn software development examples, some departments offer classes with part-time lecturers who were active in IT related companies, and some departments that encouraged collaboration with companies and active participation in programming contests.

9.3. Collaboration with Computing Qualification

There are many cases that the curriculum is designed so that many students can take the Information Technology Engineer Examination, which is the largest national examination for IT engineers. Some departments also support commercial qualification such as Microsoft Office Specialist, Microsoft Technology Associate, IT Passport, Ba-

sic Information Technician, Network Specialist, CCNA, Web Creator, LPIC, MOS etc. They introduce part to class contents, hold special course, establish course corresponding to morning examination exemption system of basic information technology examination, and offer “IT passport exercises” in sub measure. Some conduct online exams for MOS in the on-campus PC training room, or partner with Cisco Systems Inc. to offer elective courses for acquiring CCNA (CiSCO Certified Network Associates) certification.

10. Concluding Remarks

The findings found through Study D are listed below. We think that the efforts to solve the problems 2 and 5–7 are important in the future:

1. We obtained 338 responses to survey type D for the IT education to obtain high school teacher’s license on IT “Subject” Information”. The ratio of the courses that responded to the survey D among the accredited courses that can acquire a high school subject “information” type of license is 65.3%. The ratio at national university is 70.1%, at public university is 82.4%, and at private university is 63.2% respectively.
2. 30% of the respondents in Survey D overlap with Survey type A majored in the computing discipline, but there are also the cases at which teacher training courses of subject “Information” are provided at non-IT departments. Student achievement is generally higher at programs provided at computing departments.
3. The effort for general computing education is large, but the average achievement level is not so high compared with other domains.
4. The effort for “generic skills that students studying informatics should acquire” is large, and the average achievement level for skills is also high compared to other domains.
5. The total number of student quotas in the teaching curriculum of the subject “Information” is 20,854, but the number of enrolled students in the teacher training course in the first academic year is 5,006, and the number of license holders is only 369. Many students leave on the way because the number of teachers employed in the subject “information” is extremely small.
6. Teachers of the subject “information” remain as 198 students as a course of teaching professional graduates (including teacher training courses other than “information”) in FY2007. There are 1,290 high school teachers other than “information”. There are 3,173 students going on to graduate school, 11,371 are employees, 1,332 are unknown. The students acquiring a license for the subject “information” secured the competitiveness at the time of hiring teachers by acquiring multiple licenses.
7. 31.4% (106 cases) of the departments and courses do not provide educational computer systems at the university.

The data collected in this survey is useful for understanding the detailed status at the timing immediately before the re-accreditation of the teacher's courses scheduled to start from 2020. In the future, it is desirable to conduct another survey after the re-accreditation in order to analyze the difference. By analyzing other survey data such as survey C, it can be expected to clarify the characteristics of educational contents of "operation of information equipment".

For the teacher training program of the subject "Information", there are provisions of Article 5 of the Education Employee License Law Enforcement Regulations and Article 66-6 by the ministry of education in Japan (Operation of Information Equipment). However, the specific curriculum design is left to independent departments. Based on the formulation of "reference standards for informatics", it is expected that future curriculum standards for the subject "information" will be presented in a manner that is associated with the same reference standard.

Acknowledgment

This research is supported by JSPS KAKENHI Grant Numbers 16K01022 and 17K01036 as well as by the Ministry of Education, Culture, Sports, Science and Technology, Japan. The authors greatly appreciate the faculty members and the administration officers of the universities who take time to respond to our survey.

References

- Hagiya, M. (2015). Defining informatics across Bun-kei and Ri-kei, *Journal of Information Processing*, 23(4), 525–530.
- IPJSJ (2018). *Computing Curriculum Standard J17*, Information Processing Society of Japan. (In Japanese). Available at https://www.ipsj.or.jp/annai/committee/education/j07/curriculum_j17.html
- Kakeshita, N., Kakeshita, T. (2017). National survey of Japanese universities on IT education: analysis of general computing education. In: *Proc. 12-th Int. Conf. on Digital Information Management (ICDIM 2017)*, 104–109.
- Kakeshita, T. (2017a). National survey of Japanese universities on IT education: analysis of non-IT departments and courses. In: *Proc. 12-th Int. Conf. on Digital Information Management (ICDIM 2017)*, 81–86, 2017.
- Kakeshita, T. (2017b). National survey of Japanese universities on IT education: overview of the entire project and preliminary analysis. In: *Proc. Int. Conf. on Computer Supported Education (CSEDU 2017)*, 607–618.
- Kakeshita, T. (2018). National survey of Japanese universities on computing education: Analysis of departments majored in computing discipline. *Olympiads in Informatics*, 12, 69–84. DOI: 10.15388/oi.2018.06
- Kakeshita, T., Ohtsuki, M. (2011). A web-based survey system to analyze outcomes and requirements: a case for college level education and professional development in ICT. In: *Proc. 5-th Int. Conf. on Society, Cybernetics and Informatics (IMSCI 2011)*, 82–87.
- Kawamura, K. (2008). Computing curriculum standard J07: computing in general education. *IPJSJ Magazine*, 49(7), 768–774. (In Japanese)
- MECSST (2017). *FY2016 School Basic Survey*. Ministry of Education, Culture, Sports, Science and Technology (MECSST). (In Japanese)
- Ohtsuki, M., Kakeshita, T., Takasaki, M. (2017) National survey of Japanese universities on IT education: analysis of educational computer system. In: *Proc. 12-th Int. Conf. on Digital Information Management (ICDIM 2017)*, 98–103.



K. Sumi is a professor at Faculty of Education, Saga University, Japan. He received his Ph.D. degree in school education from Hyogo University of Teacher Education, Japan in 2014. His major is teaching method of information technology or technology education. He is a member of Information Processing Society of Japan (IPSJ) and The Japan Society of Technology Education (JSTE).



M. Ohtsuki is a senior lecturer at Computing Division, Saga University, Japan. She received her Ph.D. from Kyushu University in 1999. Her major research interests include computer aided ICT education, and software development methodologies including software testing. She is a committee member of JaSST (Japan Symposium on Software Testing) in Tokyo and is a commissioner at ASTER (Association of Software Test EngineerRing). She published several books about software development tools such as CVS, CppUnit etc.



T. Kakeshita is an associate professor at Computing Division, Saga University, Japan. He received his Ph.D. degree in Computer Science from Kyushu University, Japan in 1989. His major research interests include quantitative analysis of ICT education and ICT certification, and complexity analysis of database and software systems. He received an excellent educator award from Information Processing Society of Japan (IPSJ) in 2013. He joined many activities such as IPSJ educational activity, Certified IT Professional Certificate (CITP), accreditation at Japan Accreditation Board for Engineering Education (JABEE) and ISO standard development (ISO/IEC JTC1/SC7/WG20).

Analyzing Task Difficulty in a Bebras Contest Using Cuttle

Willem van der VEGT¹, Eljakim SCHRIJVERS²

¹*Dutch Olympiad in Informatics, Windesheim University for Applied Sciences
PO Box 10090, 8000 GB Zwolle, The Netherlands*

²*Dutch Olympiad in Informatics, Eljakim IT
PO Box 85183, 3508 AD Utrecht, The Netherlands
e-mail: w.van.der.vegt@windesheim.nl , eljakim@cuttle.org*

Abstract. Predicting the difficulty level of a task on the concepts of computer science or computational thinking, like in the Bebras Challenge, proves to be really hard. But the announced difficulty level is needed in the contest format used in many local challenges. The Dutch contest system Cuttle has a new module for analysis. This is applied to one specific contest in order to find parameters explaining task difficulty. Using quantitative methods we were able to confirm a relation between answer types and difficulty and a tendency that tasks on data, data structures and representation are better answered than tasks on algorithms and programming.

Keywords: Bebras contest, answer types, question difficulty, P-value, Rit-value, contest system.

1. Introduction

Founded in 2005 in Lithuania, Bebras has developed into an annual International Challenge on Informatics and Computational Thinking amongst the young (Bebras, 2019). In 2018 students from over fifty countries compete in their national contest. The questions used in these challenges are mostly chosen from a common task pool, which is composed during the annual Bebras Workshop where most of the contributing countries participate. The questions are formulated in a way that no prior knowledge is required.

The contest is about computer science and computational thinking; most of the tasks are categorized as ALP: Algorithms and Programming or DSR: Data, Data Structures, and Representations. A few tasks fit in the other three categories; CPH: Computer Processes and Hardware, COM: Communications and Networking or ISS: Interactions, Systems, and Society, based on (Dagienė and Sentance, 2016). Criteria for good Bebras tasks, using a former system for classification, have been formulated by Dagienė and Futchek (2008). Dagienė and Sturupienė (2016) give an overview of current research on Bebras.

Contestants compete in their own age division. In the Netherlands we offer the challenge in the form of a contest. In the first round contestants have 40 minutes to complete 15 tasks. Tasks can have one of three answer types: multiple choice, open ended or interactive. The contest runs during one week for five different ages groups. Some countries will also have an event for the youngest age group, 6–8 years; the Dutch contest starts with grade 3; contestants are usually aged between 8 and 18 years. The best performing contestants for the four highest age divisions are invited to a university for a second round (Beverwedstrijd, 2019).

Within the contest we present tasks to the contestants as easy, medium or hard. But in practice our own classification breaks down. Earlier (Van der Vegt, 2018) we discussed ways to predict the difficulty level of specific Bebras tasks. We applied these models to the 2017 contest for the highest age group in the Netherlands. But since using the questionnaires for predicting task difficulty is very time consuming, we want to identify a few properties of a task that can be of use in predicting task difficulty. This could be helpful for the entire Bebras community, for in a lot of national challenges the announced difficulty level of a task is part of the design.

The Cuttle contest system is developed for organizing the Bebras contest in the Netherlands. This system is used in over thirty Bebras and other scientific contests. Recently it has been extended with an analysis tool. We will use this tool to investigate aspects of the tasks in a specific contest and we try to discover a relation between properties of a task and the actual difficulty of it. In this paper we will analyze the 2017 contest for the highest age group in the Netherlands, making use of the Cuttle-tool for analysis, and develop some recommendations for possible future research. We will focus on categories of tasks and answer types.

Summarizing, we will try to answer two questions: Is it possible to use the Cuttle system to collect data that can be useful for analyzing task difficulty in Bebras? And can we formulate questions for future research, based on the findings using Cuttle?

In section 2 we will give a short summary of earlier research on predicting task difficulty. Section 3 will describe the selection process to compose the contest, characteristics of the task set and the way the task proposals were developed before, at and after the Bebras Workshop. In section 4 we give an analysis of the overall results for the contest and we will look into detail to several properties of the tasks in the contest. Finally, we give some conclusions and a few ideas for a possible research agenda in section 5.

2. Task Difficulty

Since the core of a Bebras task is answering a question, we give an brief overview of research on question difficulty, focused on Bebras and similar tasks..

Lonati, Malchiodi, Monga and Morpurgo (2017) distinguish two main kinds of difficulties: on the one side intrinsic with the task, related to its content, and on the other side surface difficulties, depending on the task format and linguistic, structural and visual aspects.

Ahmed and Pollitt (1999) distinguish three kinds of difficulties in questions. Cognitive difficulty has to do with the concepts that are used in a question. The level of abstraction

of these concepts will determine this difficulty. Question difficulty is connected with the linguistic and structural properties of a question. Process difficulty is about the difficulty of the cognitive operations and the degree in which they use cognitive resources.

Leong (2006) makes a similar distinction; he considers content difficulty, depending of the subject matter being assessed, stimulus difficulty, related to comprehending words and phrases in a test item and accompanying information, and task difficulty, referring to the work needed to formulate or discover the answer to the question.

Several questionnaires or rubrics have been proposed to predict the difficulty of a task (Van der Vegt, 2018); these instruments each try to assign proper weights to the expected difficulty on content, stimulus and task performance, in different ratios. Some items are easy to measure. Dhillon (2003) for instance states that the number of components of a question and the number of times these components have to be repeated have a high impact on the difficulty level. Estimating the number of steps to perform a task is possible for an experienced task designer. Other ways of assessing topics in these questionnaires are not yet well described.

3. Tasks

3.1. Task Selection

In the Netherlands we work together with some other countries in the selection process to compose the contests. We receive the results of the task selection from the German speaking countries, the UK and US task pool, as well as the Belgian team. We tend to reuse tasks in more than one age group in order to reduce the total number of tasks. This way we used 34 different tasks to organize a first round in 2017 for five different age groups with 12, 15, 15, 15 and 15 tasks. Of the 15 tasks that were selected for the highest age group, 9 have also been used in the same contest but for other age groups.

For each contest the difficulty level of a task is announced as easy, medium or hard. The score a contestant can achieve depends on the expected difficulty level. For an easy task, a contestant gets 6 points for a good answer and -2 for a wrong answer. For a medium task these numbers are 9 and -3 and for a hard task 12 and -4. The original rationale behind this was that the expected score for a task when guessing should be 0. This holds only for multiple choice question with four alternatives but we have kept this scheme also for the other types of answers, open ended and interactive. If a question stays unanswered, no points are added or subtracted. To prevent negative score in case someone has only wrong answers, we start for each contestant with an initial score of 45 points.

3.2. Task Properties

All tasks were taken from the international Bebras task pool 2017, developed at the Bebras Workshop in Brescia. All tasks are proposed by one of the member countries,

after which they are reviewed in the preparation weeks before the workshop. During the workshop all tasks are discussed and improved. After the workshop tasks are translated and sometimes changed in order to fit into a national contest format. It is also possible that the answer type is altered in order to make the task easier or more difficult.

In this section we investigate three aspects of a task: category, answer type and difficulty level.

3.3. Categories

In 2017 the Bebras community has introduced five categories for tasks, based on Dagiėnė and Sentence (2016):

- ALP: Algorithms and Programming
- DSR: Data, Data Structures, and Representations
- CPH: Computer Processes and Hardware
- COM: Communications and Networking
- ISS: Interactions, Systems, and Society

In the Bebras task pool these categories are not mandatory. Table 1 shows the suggested category for each task, a short description of the task, without the background story. Even though there are several tasks about graphs or on the assignment problem, the differences between these proposals are large enough to justify the use of all these tasks within one contest.

Only for 7 of the 15 tasks a domain was proposed by the original author. For the other tasks we did our own attribution and noted it in Table 1 between brackets. Most of the

Table 1
Categories, CS topics and answer types

Task-ID	Category	Computer Science Topic	Answer type
2017-CA-12	DSR	Dynamic programming	Multiple Choice Text
2017-IS-01	ALP	Sequence, binary system	Multiple Choice Text
2017-BE-05	(ALP/DSR)	A path in a graph	Multiple Choice Images
2017-RU-03	DSR	Gray code	Interactive
2017-IR-07	COM/ISS/ALP	Search in social network graph	Multiple Choice Text
2017-CA-07	ALP	Assignment problem	Interactive
2017-PL-02	(ALP/DSR)	Levenshtein distance	Open Ended Integer
2017-CH-01b	(ALP)	Programming in a maze	Interactive
2017-CZ-04c	ALP	A path in a graph	Interactive
2017-CH-07b	(ALP/DSR)	Maximum flow problem	Open Ended Integer
2017-KR-07	(DSR)	Image compression	Multiple Choice Text
2017-SK-12a	(ALP)	Turing machine	Multiple Choice Images
2017-UK-04	ALP	Assignment problem	Multiple Choice Text
2017-KR-03	(ALP)	Optimization, scheduling	Open Ended Text
2017-SI-04	(ALP/DSR)	Binary counting	Open Ended Integer

used categories are ALP (80%) and DSR (47%). Only one task was announced as a task both on ISS and on COM (both 7% of the tasks). The category CPH was never used.

3.4. Answer Types

Within the contest we used five different answer type:

- Multiple Choice Text means the classical form with four alternatives (33 %).
- Multiple Choice Images is somewhat similar; the alternatives are now presented as images (13%).
- Open Ended Integer asks the user to input a number (20 %).
- Open Ended Text ask the user to input a string (7 %).
- Interactive means the user has to perform some kind of action to solve the problem; a grader program checks the solution (27 %).

3.5. Task Difficulty

Due to the contest format we need to identify the difficulty level of each task, or to compare the tasks with each other. There are several problems in predicting difficulty level (Van der Vegt, 2013) and last year we experimented with several tools to help in this process (Van der Vegt, 2018). For the tasks in the 2017 contest we looked at the original task proposals, the tasks in the task pool and we made of course our own assessment. This is summarized in Table 2.

Table 2
Task difficulty estimations

Task-ID	Original difficulty level	Workshop difficulty level	Contest difficulty level
2017-CA-12	III-easy	V-medium	VI-easy
2017-IS-01	V-medium	V-hard	VI-easy
2017-BE-05	IV-medium	IV-medium	VI-easy
2017-RU-03	II-medium	IV-easy	VI-easy
2017-IR-07	IV-easy	V-medium	VI-easy
2017-CA-07	V-hard	V-medium	VI-medium
2017-PL-02	V-hard	V-hard	VI-medium
2017-CH-01b	IV-easy	V-medium	VI-medium
2017-CZ-04c	V-medium	V-hard	VI-medium
2017-CH-07b	VI-hard	V-hard	VI-medium
2017-KR-07	IV-medium	VI-hard	VI-hard
2017-SK-12a	VI-medium	VI-hard	VI-hard
2017-UK-04	VI-hard	VI-hard	VI-hard
2017-KR-03	VI-medium	VI-hard	VI-hard
2017-SI-04	V-medium	V-medium	VI-hard

3.6. The Cuttle Contest System

The Cuttle system is evolved from the system build for the first Bebras contests in the Netherlands. Since the early start in 2006 we have organized 43 contests, usually two round per year and some demonstration games. Within the system 723 Dutch tasks are available. Each task can get a difficulty level per age group, a category can be assigned and it is also possible to indicate a CS Skill: Abstraction, Algorithmic Thinking, Decomposition, Evaluation and Generalization. The system is also available in other languages.

In 2018 20 countries organized their national Bebras challenge with the Cuttle system: Australia, Austria, Canada, Germany, Greece, Hong Kong, Iceland, India, Ireland, Japan, Malaysia, New Zealand, Netherlands, Norway, Romania, South Africa, Switzerland, Thailand, United States and the UK, with in total almost one million contestants. The system is also used for several other Bebras-like contests.

4. Results

In this section we will apply the new Analytics part of the Cuttle-system on the first round of the 2017 contest in the Netherlands for the highest age group (16–18 years). This contest had 1621 participants. The data we study are the results of this contest for these participants. We look at the correct answers, at the fraction of the participants not answering a specific task. And we try to relate some of the numbers to the theory on question difficulty.

In Fig. 1 the number of well-answered tasks is shown (max. 15) as well as the distribution of the scores (max. 180). The distribution patterns of both correct answers and scores appear to resemble the normal distribution.

The contest system also provides general data, like the ones shown in Fig. 2. With a maximum score of 180 the whole range from 0 to 180 turned out to be possible, with an average of 92.9 and a standard deviation of 32.4.

The system gives also the detailed scores for all tasks. Table 3 shows a part of the output, focusing on a few major measures. P_{all} is the percentage of correct answers across all participants; this P-value is often used as an indication of the difficulty level (Van der Vegt, 2013). Goldebeld (1992) states that in an ideal exam all P-values should be between 30 and 70%; but since Bebras is not an exam but a challenge, we value this not as an important restriction for our tasks. The R_{it} gives the correlation between the score of a task and the overall score as a percentage. An R_{it} -score of 40% or above is seen as an indication that the task was very good fitting in a contest (Goldeberg, 1992). An R_{it} -score below 20% indicates an atypical result for a task; if such a score occurs it means you will have to investigate if there is a problem with the task. In this perspective our outcomes were very satisfying.

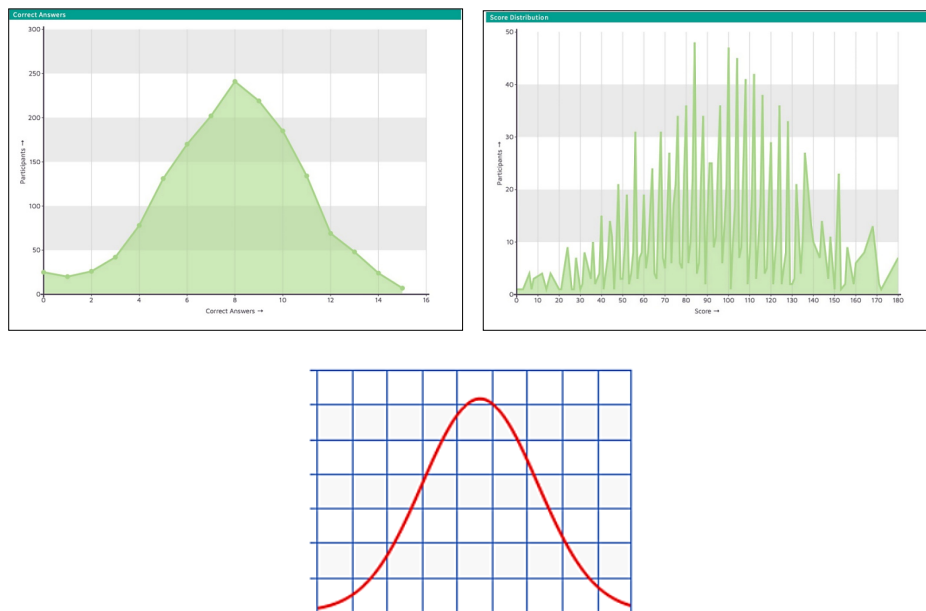


Fig. 1. Number of correct answers and score distribution, compared with a normal distribution.

Group Analytics	
analytics score lowest	0
analytics score highest	180
analytics score average	92.9
analytics score stdev	32.4
analytics relative stdev	0.3
analytics max possible	180
analytic avg p score	51.6
analytics participants	1621

Fig. 2. General analytics.

Table 3
Details per task

Task	P _{all}	R _{it}	%NA	Task	P _{all}	R _{it}	%NA
2017-CA-12	87.4	33.5	2.04	2017-CZ-04c	45.2	55.8	12.16
2017-IS-01	86.4	44.9	3.21	2017-CH-07b	16.5	34.2	11.79
2017-BE-05	81.7	37.5	2.10	2017-KR-07	48.4	57.3	20.37
2017-RU-03	65.7	42.8	6.85	2017-SK-12a	43.1	52.5	18.64
2017-IR-07	41.4	39.1	2.53	2017-UK-04	35.2	40.1	23.09
2017-CA-07	75.9	38.6	15.93	2017-KR-03	15.7	45.5	32.10
2017-PL-02	68.1	46.0	6.05	2017-SI-04	10.1	38.8	23.77
2017-CH-01b	63.8	41.3	23.95				

4.1. Specific Task Details

The Cuttle contest system allows us to analyze the results of a task in more details. Fig. 3 shows the plots of the two tasks with the lowest and the highest R_{it} -score. The five values in the graph are the P-values for five different percentiles. So the lines will need to be ascending or at least not-decreasing. The low R_{it} -value in the left graph can be recognized as a bend line, where the highest line in the left graph approximates a straight line indicating a high R_{it} -value. The lower line in this graph is for a younger age group. The graph shows that for the best performing contestants in both age groups the P-values are almost similar; but the differences between age groups for less well performing contestants are much more age dependent.

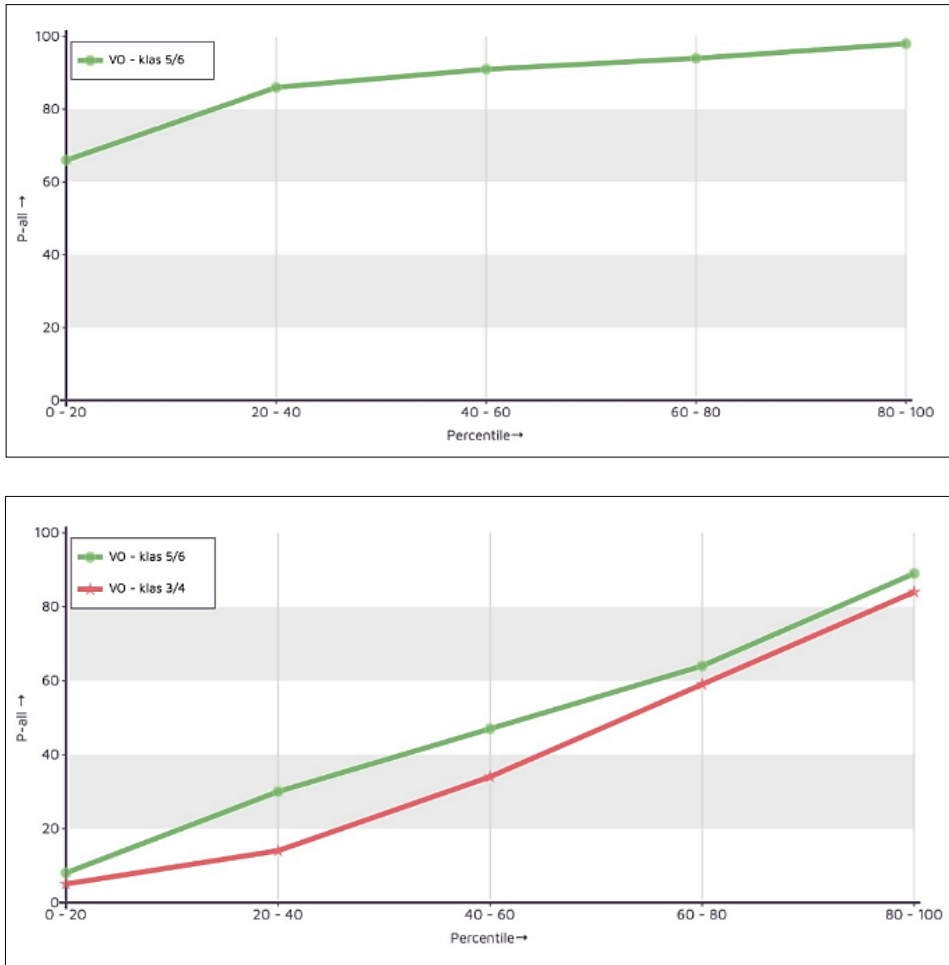


Fig. 3. The results of task 2017-CA-12 and task 2017-KR-07.

4.2. Categories

Using the output of the content system, we investigated the results for different categories. We looked at DSR and ALP; we will qualify the result for the one task that combined ALP, ISS and COM as a task on ALP. Though the numbers are small, there seems to be a tendency that DSR-tasks have shown to be a bit less difficult. And the combination of ALP and DSR is harder than the sole categories. This suggests that tasks on ALP require a higher cognitive load and combining both categories increases it even more. Performing an algorithm requires to make more steps in your memory or to use external memory like paper and pencil. That makes the solution process more error-prone. Another way to look at it is that DSR-tasks are more of a static nature while ALP-tasks are more dynamic. According to Leong (2006) increasing or decreasing the number of steps needed to find a solution influences task difficulty. It is interesting to investigate whether the nature of ALP-tasks makes it harder to reduce the number of steps in the solution process.

The P-value, the R_{it} and the percentages of non-answered tasks for categories are shown.

4.3. Answer Type

The same approach is used for analyzing the results for the different answer types. Table 5 shows the five different answer types as described in Section 3.2. As was expected, the Open Ended tasks turned out to be the hardest. The Open Ended Text task had almost one third of the participants not answering. This result can be attributed to the much larger search space in these open tasks, increasing task difficulty.

Table 4
Results per category

Categories	n	P_{all}	R_{it}	%NA
ALP/DSR	4	45.5	40.8	13.01
Only ALP	8	50.1	43.9	15.41
Only DSR	3	67.2	44.5	9.75

Table 5
Results per answer type

Answer type	n	Pall	Rit	%NA
MC Text	5	59.8	43.0	10.25
MC Images	2	62.4	45.0	10.37
Open Ended Integer	3	31.6	39.7	13.87
Open Ended Text	1	15.7	45.5	32.10
Interactive	4	62.7	44.6	14.72

4.4. Task Difficulty

An open question for us is whether it is possible to make one scale for difficulty level and age group. In practice we use the assumption that the difficulty level of a task is reduced one step if you offer the task to the next higher age group. This way we can for instance offer the same task as hard for age group IV, as medium for age group V and as easy for age group VI. The results for a task for adjacent age groups can turn out to be really different, due to the computer science concepts in it or the cognitive development of the contestants of a specific age. Table 6 presents the P-values of tasks that were used in several age groups. The average difference of the P-values of age group VI and age group V is 12.8 percent, the difference between age groups VI and IV is 25.2 percent and for the two tasks that were also in the contest for age group III the difference was 42.3 percent.

An interesting research question would be to look for an explanation of the small differences in difficulty level for the one task, for instance 2017-CH-01b, and the large difference for some other tasks like 2017-CZ-04c. Understanding these differences would really help us to predict the difficulty level of a Bebras task for a certain age group.

5. Conclusions

We have tried to answer two questions. It is possible to use the Cuttle system to collect data that can be useful for analyzing task difficulty in Bebras? The new features of Cuttle offered us the chance to investigate the results of a contest in a much more detailed way. We were able to check and confirm that the contest we analyzed had a proper correlation between the individual tasks and the contest as a whole, we could reflect on the actual difficulty level and compare it to the announced difficulty.

And can we formulate questions for future research, based on the findings using Cuttle? We were able to show the relation between answer type and the P-values of the

Table 6
P-values for other age groups

	III	IV	V	VI
2017-IS-01			69.0	86.4
2017-RU-03	22.8	35.7	53.8	65.7
2017-CA-07			61.8	75.9
2017-PL-02		34.9	49.3	68.1
2017-CH-01b		51.7	62.0	63.8
2017-CZ-04c	3.5	6.9	20.5	45.2
2017-CH-07b		4.1	9.2	16.5
2017-KR-07			39.9	48.4
2017-SK-12a			32.6	43.1

tasks. This is in line with earlier results on question difficulty, so answer type is useful as a parameter on task difficulty. We also showed that at least in this contest tasks of category DSR seems to be more easy than ALP tasks while combining these category increases the difficult even further. Repeating this analysis for other contests is needed to check if this reveals a general pattern and if category can be a parameter in predicting task difficulty. We found an average increase of around 13% in P-values for the same task used in the next higher age group. But there are larger differences between tasks and it will be interesting to look into these differences in order to be able to predict the difficulty level for each specific age group. The new tool for analysis can help us in this future research.

References

- Ahmed, A., Pollitt, A. (1999). Curriculum Demands and Question Difficulty. Paper presented at IAEA Conference, Slovenia, May.
- Bebras website (2019). <http://bebras.org/>
- Beverwedstrijd (2019). <http://www.beverwedstrijd.nl/> (in Dutch)
- Dagienė, V., Futschek, G. (2008). Bebras international contest on informatics and computer literacy: Criteria for good tasks. In: R.T. Mittermeier and M.M. Syslo (Eds.), *ISSEP 2008, LNCS 5090*. Springer-Verlag Berlin Heidelberg, pp. 19–30.
- Dagienė, V., Sentance, S. (2016, October). It's computational thinking! Bebras tasks in the curriculum. In: *International Conference on Informatics in Schools: Situation, Evolution, and Perspectives*. Springer International Publishing, pp. 28–39.
- Dagienė, V., Sturupienė, G. (2016). Bebras – a sustainable community building model for the concept based learning of informatics and computational thinking. *Informatics in Education*, 15(1), 25–44.
- Dhillon, D. (2003). *Predictive Models of Question Difficulty – A Critical Review of the Literature*. Manchester, AQA Centre for Education Research and Policy
- Goldebel, P. (1992). *Toets en – Itemanalyse Met TIA*. Cito, Arnhem. (in Dutch)
- Leong, S.C. (2006). On varying the difficulty of test items. Paper presented at the 32nd Annual Conference of the International Association for Educational Assessment, Singapore.
- Lonati, V., Malchiodi, D., Monga, M., Morpurgo, A. (2017). How presentation affects the difficulty of computational thinking tasks: an IRT analysis. In: *Proceedings: 17th Koli Calling Conference on Computing Education Research: Koli Calling 2017: November 16–19, 2017: Koli, Finland*. ACM, 60–69.
- Van der Vegt, W. (2013). Predicting the difficulty level of a Bebras task. *Olympiads in Informatics*, 7, 132–139.
- Van der Vegt, W. (2018). How hard will this task be? Developments in analyzing and predicting question difficulty in the Bebras Challenge. *Olympiads in Informatics*, 12, 119–132.



W. van der Vegt is teacher's trainer in mathematics and computer science at Windesheim University for Applied Sciences in Zwolle, the Netherlands. He is one of the organizers of the Dutch Olympiad in Informatics and he joined the International Olympiad in Informatics since 1992. He has been a part of the international Bebras community from the start in 2005 and has been a member of the Bebras board, with a specific interest in task development.



E. Schrijvers is chair of the Dutch Foundation of the Informatics Olympiad. Since 1994 he is teamleader of the Netherlands at the International Olympiad in Informatics. He runs Eljakim IT, which develops and maintains Cuttle, the contest system that is used for Bebras in the Netherlands. This system is used in over thirty Bebras and other scientific contests.

Programming, Software Development, and Computer Science – The Golden Triangle

Tom VERHOEFF

*Department of Mathematics and Computer Science
Eindhoven University of Technology
e-mail: t.verhoeff@tue.nl*

Abstract. I present my thoughts on programming, software development, and computer science (CS), and their inevitable relationship. Originally this was intended to help prepare some CS courses aimed (also) at non-CS university students. But it is also relevant for students in secondary education, especially if they have an interest in participating in the International Olympiad in Informatics.

Keywords: Computer science, programming, software engineering, education, competition.

1. Introduction

The reason for writing this note is my involvement in defining some computer science (CS) courses at Eindhoven University of Technology in the Netherlands, especially for non-CS students. Here are some initial questions to set the scene.

- What should everyone know about computer science?
- What should every university student know about computer science?
- What should every engineering student know about computer science?

With “know about” I do not just mean superficial meta-knowledge, where someone knows of the existence of some CS topics and the related jargon without knowing any actual content. That is, the questions could be rephrased as

What CS knowledge and skills should every . . . acquire?

Compare this to similar questions for mathematics, physics, chemistry, biology, etc. The founders of the International Olympiad in Informatics (IOI) must have asked such a question as well. Note that the three questions are related but do not have the same answers.

Related to this question, one should also ask *why* such knowledge and skills are relevant. Subsequent questions are:

- When to teach CS? Earlier or later?
- How to teach CS? Integrated in the student’s primary domain of interest, or more purely? According to what didactic principles?

An important reference document is ACM/IEEE CS Curriculum (2013), which covers most, if not all, of the topics that I touch here. But it weighs in at over 500 pages, and in many cases it only presents alternatives and tradeoffs, without making specific choices or recommendations. We will not answer all questions; neither will we make definite choices or recommendation. But we will delve a bit deeper into these issues. (Short answer: my opinions come close to Kernighan (2017); see below for details.)

2. What and Why

Let's start with the why. Why would someone need to have CS knowledge and skills? Here are some answers.

1. Because our world has become so much more computational in recent decades.

It is important to know about CS simply in order to “survive”; without that knowledge, life will be more difficult. All kinds of decisions that people need to make involve computers, automation, and cyberspace. We can communicate data to any place on earth, store all data that we collect, and process data anywhere we like (also see Verhoeff (2013)). There are many digital dangers nowadays, not in the least due to the rise of artificial intelligence (AI) driven by big data (I recommend du Sautoy (2019) for an interesting exploration).

2. Because in both professional and personal life, people (scientists and engineers, but also entrepreneurs, and anyone involved with information and its processing) will be required to apply some CS knowledge and skills.

Creating a spreadsheet with formulae, developing software tools to help create (non-CS) products, leading a multidisciplinary team to design and develop products that include domain-specific software, running virtual experiments and analyzing the results, formulating computational models, communicating domain knowledge to software developers. Here are some diverse ways in which program code plays an important role:

- To create products (such as models for 3D printing and pictures using computer graphics).
- To operate devices (such as cars, drones, and robots).
- To provide services on the web (such as interactive maps and secure email).
- To solve computational problems (such as optimal routing of packages and aircraft, weather forecasting).
- To create software tools that help develop software (such as compilers, interpreters).
- To analyze massive amounts of data (so as to rank web pages by relevance, and discover new medicines).

3. Because it is interesting,

Just as any other science can be interesting, CS is a very interesting discipline, with important relationships to mathematics, physics, chemistry, biology, psychology, economics, etc.

When addressing the what, it is useful to make the following distinctions. We follow Computing at School Working Group (2012).

Digital Literacy (DL) “the ability to use computer systems confidently and effectively, including:

- Basic keyboard and mouse skills.
- Simple use of ‘office applications’ such as word processing, presentations and spreadsheets.
- Use of the Internet, including browsing, searching and creating content for the Web, communication and collaboration via e-mail, social networks, collaborative workspace and discussion forums.”

Information Technology (IT) “the creative and productive use and application of computer systems, especially in organisations, including considerations of e-safety, privacy, ethics, and intellectual property.”

Computer Science (CS) “the study of the foundational principles and practices of computation and computational thinking, and their application in the design and development of computer systems.”

We will presume that our students are digitally literate, and that DL is not a goal of our courses (nor of the IOI). Although IT is important, we should not include it as goal of our courses, because IT is focused more on short-term technological issues. The principles that underly IT systems are long lasting, and belong to CS.

Peter Denning provides a broad classification of CS principles in Denning (2003):

Computation “meaning and limits of computation”

Communication “reliable data transmission”

Coordination “cooperation among networked entities”

Recollection “storage and retrieval of information”

Automation “meaning and limits of automation”¹

Evaluation “performance prediction and capacity planning”

Design “building reliable software systems”

Recently, several books have appeared that put computer science in a broader perspective: Rosenbloom (2013); St. Amant (2012); Tedre (2014). Also the Advanced Placement (AP) Computer Science course is turning to an approach through principles.²

Brian Kernighan (2017) (original article Kernighan (2008); first edition Kernighan (2011)) has the subtitle ‘What you need to know about computers, the Internet, privacy, and security’, and is summarized on Amazon.com as follows.

¹ In Denning and Martell (2015), automation is dropped as a separate category.

² <http://apcsprinciples.org>

“[This book] explains how computer hardware, software, networks, and systems work. Topics include how computers are built and how they compute; what programming is and why it is difficult; how the Internet and the web operate; and how all of these affect our security, privacy, property, and other important social, political, and economic issues. This book also touches on fundamental ideas from computer science and some of the inherent limitations of computers.”

2.1. Programming

According to the DL-IT-CS definitions above, *programming* can belong both to IT (“application of computer[ized] systems”) and CS (“development of computer[ized] systems”). The former concerns the more concrete side of programming, whereas the latter focuses more on abstract aspects, such as design.

As argued by Denning in Denning (2004), “The persistent public image of computing as a field of programmers has become a costly myth. Reversing it is possible but not easy.” It is also instructive to consult the online FAQ of Denning (2003), and read the questions and answers about programming.

In Denning’s classification, programming does not appear as a principle; instead, he treats it as a practice. Denning and Martell Denning and Martell (2015) have this to say about programming.

- “What is the paradigm of computing? . . . There were three waves of attempts to unify views. . . . The first . . . argued that computing was unique among all the sciences in its study of information processes. A catchphrase of this wave was that “computing is the study of phenomena surrounding computers.” . . . The second wave focused on programming, the art of designing algorithms that produced useful information processes. . . . A catchphrase of this wave was “computer science equals programming.” In recent times, this view has foundered because the field has expanded well beyond programming and because public understanding of a programmer became so narrow (a coder). . . . The third wave . . . defined computation as the automation of information processes in engineering, science, and business. Its catchphrase was “computing is the automation of information processes.”” Denning and Martell (2015, Ch. 1 (Computing))
- “A *program* is a set of instructions arranged in a pattern that causes the desired function to be calculated. *Programming* is the art of designing a program and providing convincing evidence that the program computes its function correctly. A *computing system* is a combination of program and machine.” Denning and Martell (2015, Ch. 4 (Machines))
- “A *program* is an expression of an algorithm, encoded for execution on a machine. . . . Programming is the practice of encoding algorithms for execution on a machine.” Denning and Martell (2015, Ch. 5 (Programming))

- “[I]t appears to many that algorithm analysis and programming are the heart of computer science. This conclusion does not seem right to us. . . . [I]t appears to us that the architecture of computers is as important as the algorithms they run. This is abundantly evident in the principles of computing. Many principles are about the systems on which computations run. We cannot give a complete picture of computing if we limit our principles to algorithms and ignore the principles of architecture.” Denning and Martell (2015, Ch. 12 (Afterword))

By the way, their book does not seem to contain an explicit definition of *algorithm*.

Robert St. Amant has this to say in St. Amant (2012, Ch. 5 (Programming: Putting Plans into Action)):

“Algorithms and collections of information, organized by abstract data types, need to be translated into a form that a computer can process. This is what programs are for: they translate between the abstract and the concrete.

“*Programming* means expressing abstractions in a language that a computer can deal with. Given what we know about computer architecture . . . I suspect programming may sound a bit daunting. And it can be, . . .”

Kernighan (2017) devotes

- Chapter 4 to algorithms (a dozen pages):

“. . . algorithms, which are abstract or idealized descriptions of processes that ignore practicalities. An algorithm is a precise and unambiguous recipe. It’s expressed in terms of a fixed set of basic operations whose meanings are completely known and specified; it spells out a sequence of steps using those operations, with all possible situations covered; and it’s guaranteed to stop eventually.” (First paragraph of Ch.5)

- Chapter 5 to programming and programming languages (twenty pages):

“. . . , a *program* is anything but abstract – it’s a concrete statement of every step that a real computer must perform to accomplish a task.” (second paragraph of Ch.5)

- Chapter 7 to learning to program (in JavaScript, a dozen pages), including loops, conditionals, libraries and interfaces:

“I think it’s important for a well-informed person to know something about programming, perhaps only that it can be surprisingly difficult to get very simple programs working properly.” (First paragraph of Ch.7)

In Barr *et al.* (2010), the issue of ‘What everyone needs to know about computation’ is discussed by four panelists. One conclusion appears to be that some form of programming (in a language with a well-defined semantics; hence, executable) is necessary, if only to keep people from becoming sloppy in expressing their computational ideas. By the way, this has also been the motivation to include actual programming in the International Olympiad in Informatics (IOI, 2019). The IOI is an algorithmic problem solving contest for high school students, aimed at identifying, encouraging, and challenging students with a talent for CS. The contestants are required to solve *algorithmic problems*, and code their solutions in one of the supported programming languages. These programs are then checked by execution.

Chris Granger argues in Granger (2015) that ‘coding is not the new literacy’, but that modeling is. Others have countered that modeling must be done in some language, and that in the end this comes pretty close to programming.

2.2. Software Development

When programming is done professionally,

- with the goal of developing complex software products,
- often as part of even more complex systems,
- that are maintainable over many years,
- intended for external non-CS customers and users,
- involving multidisciplinary development teams,
- under economic resource constraints,

a whole set of new problems arises. The field of *Software Engineering* (SE) attempts to address these problems. It goes well beyond the basics of programming, including

- domain engineering and requirements engineering;
- modeling;
- architecture;
- evolution, maintenance;
- dealing with errors, quality control, validation & verification, reviewing & testing;
- configuration management, revision control;
- project management;
- specialized software tools for these.

However, anyone doing serious programming, even if only for personal use, can benefit from the key insights of software engineering. In fact, as a teacher you do someone a disservice by not explaining these insights, because without them, programming can be a frustrating experience. In particular, the topics of (1) dealing with errors (in a broad sense, including unit testing), (2) configuration management (e.g., using Git), and (3) coding idioms, design patterns, and architecture are essential. It surprises me that the IOI environment still does not offer standard tools for (unit) testing and configuration management, given that the contestants must develop code that works.

3. Challenges

The Asian board game *go* has very simple rules, yet it is a notoriously deep game. Only recently³ have we succeeded in letting computers play above the mere beginner's level (ComputerGo – Wikipedia, 2019). In chess, the world champion has been beaten by a computer already back in 1996 (DeepBlue – Wikipedia, 2019).

Programming is like *go*: the basics are very simple, but it is notoriously hard to write good programs. Michael Jackson, the British computer scientist, captures this well in his essay “Brilliance” (Jackson, 1995), which I quote here in full.

Some years ago I spent a week giving an in-house program design course at a manufacturing company in the mid-west of the United States. On the Friday afternoon it was all over. The DP Manager, who had arranged the course and was paying for it out of his budget, asked me into his office.

‘What do you think?’ he asked. He was asking me to tell him my impressions of his operation and his staff. ‘Pretty good,’ I said. ‘You’ve got some good people there.’ Program design courses are hard work; I was very tired; and staff evaluation consultancy is charged extra. Anyway, I knew he really wanted to tell me his own thoughts.

‘What did you think of Fred?’ he asked. ‘We all think Fred is brilliant.’ ‘He’s very clever,’ I said. ‘He’s not very enthusiastic about methods, but he knows a lot about programming.’ ‘Yes,’ said the DP Manager. He swiveled round in his chair to face a huge flowchart stuck to the wall: about five large sheets of line printer paper, maybe two hundred symbols, hundreds of connecting lines. ‘Fred did that. It’s the build-up of gross pay for our weekly payroll. No one else except Fred understands it.’ His voice dropped to a reverent hush. ‘Fred tells me that he’s not sure he understands it himself.’

‘Terrific,’ I mumbled respectfully. I got the picture clearly. Fred as Frankenstein, Fred the brilliant creator of the uncontrollable monster flowchart. ‘But what about Jane?’ I said. ‘I thought Jane was very good. She picked up the program design ideas very fast.’

‘Yes,’ said the DP Manager. ‘Jane came to us with a great reputation. We thought she was going to be as brilliant as Fred. But she hasn’t really proved herself yet. We’ve given her a few problems that we thought were going to be really tough, but when she finished it turned out they weren’t really difficult at all. Most of them turned out pretty simple. She hasn’t really proved herself yet – if you see what I mean?’

I saw what he meant.

³ I wrote this sentence in 2015. In the meantime, *AlphaGo* convincingly beat the world champion *go*. And not much later *AlphaZero* thrashed *AlphaGo*.

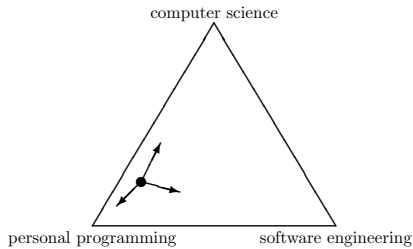


Fig. 1. The force field in which to place a CS/'programming' course.

Managers and directors (educational and industrial), administrators, politicians, they all often still hold similar misunderstandings and misconceptions.

Fig. 1 shows the three forces that need to be balanced in a computer science course on/using programming, both in setting the goals and choosing the means. The same holds for a competition like the IOI. These forces pull towards the three 'pure' topics:

Computer science teach general concepts and insights from computing.

Personal programming teach a programming language for personal use.

Software engineering teach how to develop software beyond personal use.

I call it 'personal programming' here to avoid confusion with 'professional programming' as applied in software engineering.

Note that these three goals are overlapping but quite distinct. A course and contest must be positioned in this force field, especially when no prior knowledge is presumed. Paying more attention to one aspect will detract attention from other aspects. It is true that through (personal) programming, most of Denning's great principles of computing can be visited, provided the trip is carefully planned. It is also the case that many lessons from software engineering are valuable (some would even say indispensable) when doing personal programming.

Should these topics be addressed in some particular order? It seems to make little sense to start on software engineering without a background in computer science and programming. On the other hand, one can start to address software engineering principles and practices early on. For instance, to write program code that is readable and understandable, through proper indentation, spacing, comments, naming, and structuring. One can write comments that document the interfaces of functions and classes. One can think about testing, and write unit tests.

4. Choice of Programming Language and Tools

Programming languages come in many different flavors. A programming language typically supports one or more programming *styles* or *paradigms*: structured, imperative (procedural, object-oriented), declarative (functional, logical), concurrent, parallel. Properties of programming languages that I consider relevant:

- Degree of formality (of syntax and semantics)
 - Informally described algorithms, such as in Cormen (2013);
 - More formal pseudo code, such as in Cormen *et al.* (2009);
 - Real' (machine executable) languages.
- standardized by international organization (ISO, IEEE, ECMA, ANSI),
- vendor-specific and commercial, versus open-source,
- having up-to-date and supported implementations,
- widely used in industry, versus academic,
- usable by beginners,
- with a large user community,
- for a broad range of applications (including support for GUI, graphics, database connection, internet protocols),
- with good execution performance (speed, memory),
- availability of user-friendly tools (IDE, profiling, testing, documentation generation, integrated version control),
- availability of courses and textbooks.

Here is the list of programming languages that I considered (all are higher level, general purpose, structured). They are split into two groups, and each group is roughly in chronological order. The first group (above the line), I consider serious candidates, and the second group is there mostly because others wanted me to consider them.

(Object) Pascal, Delphi From 1970s, imperative object-oriented (not pure), compiled or interpreted, ISO standard (but that is dated), simple readable syntax, strong static typing, reasonably good compilers and IDE (also open source); used to be the foundation of MacOS; kept alive by Embarcadero (Delphi is vendor specific)

C From 1970s, imperative, ISO standard (current: C18), imperative (not object-oriented), weak static typing, efficient, enforces thinking about low-level optimizations (adorned assembly language), quirky syntax, good (open-source and commercial) compilers and IDEs, favorite for embedded systems (control) not so good for beginners (memory management, robustness); is the basis of the Unix (and nowadays, Linux) operating system

Scheme From 1970s, functional (not pure), IEEE standard (current: 2008?); Lisp dialect; based on lambda calculus

Effel From 1980s, imperative object-oriented, with support for Design-by-Contract, ECMA-ISO standard (current: ECMA-367 from 2006)

Erlang From 1980s, functional and concurrent, aimed at critical high-reliability high-availability systems (hot swapping; think of telephone exchange systems that need to run without downtime)

Java From 1990s, imperative object-oriented (not pure), strong static typing, open standard, commercial owner (Oracle), via interpreter (JVM), C-like syntax, garbage collected, exception mechanism, good compilers and IDEs, favorite for Android mobile platforms, lots of textbooks

Python From 1990s, imperative object-oriented with functional features (not pure), dynamic typing (supports explicit type annotations since v3.5), open source (Python Software Foundation), interpreted, clean syntax⁴, extensive libraries, favorite for scripting and coordination

ECMAScript (better known as **JavaScript**) From 1990s, imperative object-oriented with functional features (not pure), interpreted, ECMA standard (current: ES2018); C-like syntax, dynamic typing, good interpreters (e.g., in web browsers), favorite for web programming (client side, nowadays also server side, and for mobile apps); also see Kernighan (2017); Verhoeff (2010)

C++ From 1990s, modernization of C, imperative object-oriented with functional features (not pure), ISO standard (current: C++17), efficient, good (open-source and commercial) compilers, favorite for embedded systems and high-performance computing, not so easy for beginners

Haskell From 1990s, pure functional, lazy, based on category theory

Scala From 2000s, object-oriented and functional (not pure), runs on JVM

Fortran From late 1950s, imperative with object-oriented extensions, ANSI-ISO-IEC standard (current: Fortran 2018), favorite for high-performance computing

Mathematica, Wolfram Language From late 1980s, commercial (Wolfram), focused on applying mathematics

MATLAB From 1980s, commercial (MathWorks), favorite for scientific/engineering modeling and simulation

R From late 1990s, a language and environment for statistical computing and graphics. Open source; based on S from the late 1970s.

Scratch From 2000s, block-structured visual, aimed at children

Go From late 2000s, imperative, C-like, commercial (Google), aimed at server-side networked applications

Swift From 2010s, commercial (Apple), aimed at mobile platform

Dart From 2010s, commercial (Google), ECMA standard, aimed at web, mobile, internet-of-things (IoT)

SageMath open source, alternative for Mathematica

Octave open source, alternative for MATLAB

The choice is not easy, and it is surprising that the CS community has not (yet?) been able to come up with a lingua franca (compare this to mathematics, where the situation is considerably better). This is also known in the IOI community. There are many trade-offs. If one wants to take popularity into account, then also consult TIOBE Index (2019).

⁴ My main gripe is that the symmetric = is used for assignment, a bad heritage from C.

Keep in mind that students (in engineering) should learn (about) multiple programming languages and paradigms.

Nowadays, I lean more towards a functional language. This is well explained by Simon Peyton Jones in Heath (2017): “If you want to see which features will be in mainstream programming languages tomorrow, then take a look at functional programming languages today.” The tendency is towards side-effectfree functions and immutable data, because these are so much easier to reason about and parallelize.

Since Python introduced type annotations that are supported by some of the tools (notably PyCharm⁵), it has moved up considerably in my list. When I don’t need the highest performance and don’t need 3D graphical output, Python is my preferred language. Together with the Jupyter notebook technology⁶, it provides a very productive interactive experience. For high performance programs, I prefer to generate C or C++ code from higher-level models through some higher-level domain-specific language (DSL).

5. Role of Didactics and Problem Domain

It is very important to motivate students, so that they take a deeper interest in computer science and programming, and will not consider it a triviality. Therefore, the application domain must be attractive. This is not easy in a course or competition with a diverse audience.

When teaching a course on programming, it is important to be aware of didactic issues. For instance, the looping construct in most programming languages is syntactically not so difficult, and even semantically it may be simple. But loops serve many purposes, and each requires further insights. Hence, it is important to teach the design of loops in a systematic way. Similarly, recursion is almost trivial from the language perspective, but didactically it needs a careful approach (Verhoeff, 2018).

Another essential topic in programming is that of abstraction (Verhoeff, 2011). Once the primitive building blocks of a programming language have been treated, it turns out that in real programs it is important to capture all kinds of abstractions (both on the level of data and actions). Defining and designing such abstractions is at the heart of computer science and programming.

Good study material should include:

- A textbook (preferably in interactive digital form, that can be searched).
- Exercises, with feedback; possibly on-line and automated, such as Codingbat (2019); Datacamp (2019).
- Web lectures (cf. Khan Academy, EdX, Lynda, etc.).

Certain choices (language, tools, problem domain, and didactic approach) will require an investment that makes it harder to change these choices later on. For instance, development of exercises and assignments is costly, but their details will critically depend on these choices.

⁵ <https://www.jetbrains.com/pycharm/>

⁶ <https://jupyter.org/> supporting over 40 programming languages

6. Conclusion

It is legitimate to expect that non-CS students will need some programming skills. It is also legitimate to expect that they need at least some software engineering skills. But they also need more fundamental insights in computing concepts that transcend programming and software development.

Whether this can be combined in a single course (a single competition) is debatable, and would certainly pose an extreme challenge. When covering this material in multiple courses, these courses must be carefully coordinated.

We should avoid pretending that a ‘personal programming’ course will make you a computer scientist, or a software engineer. Learning a programming language is relatively easy, writing good software is hard.

We identified three forces (see Fig. 1), and observed the following dependencies.

- Even if a course is intended solely as an introduction to CS, it is a good idea to include some programming involving an executable programming language.
Motivation: Using informal notation or pseudo code to express algorithms leaves the door open for sloppiness and misunderstanding.
Note: In this case, only a minimum of software engineering issues need to be addressed.
- If programming needs to be applied in practice, no matter on what scale, then computer science and software engineering knowledge is needed.
Motivation: Without CS and SE knowledge, it is too easy to create low-quality software. This is frustrating, time consuming, and costly.
Note: The scale of application will determine the amount of CS and SE to include.
- To understand the lessons of software engineering, it is necessary to have some programming experience.

The appendices list, what I consider, the top-10 most relevant topics from each of the three corners. Compare this to IOI Syllabus (2019).

References

- ACM/IEEE-CS Joint Task Force on Computing Curricula (2013). *Computer Science Curricula 2013*. ACM Press and IEEE Computer Society Press. DOI: [dx.doi.org/10.1145/2534860](https://doi.org/10.1145/2534860)
- Barr, J. *et al.* (2010). What everyone needs to know about computation. *SIGCSE'10*, pp.127–128, 10–13. [Codingbat, codingbat.com](http://codingbat.com)
- Cormen, Th.H. (2013). *Algorithms Unlocked*. MIT Press.
- Cormen, Th.H. *et al.* (2009). *Introduction to Algorithms* (3rd Ed.). MIT Press.
- Computing at School Working Group (2012). *A Curriculum Framework for Computer Science and Information Technology*. <https://www.computingschool.org.uk>
- Datacamp*, <https://datacamp.com>
- Denning, P.J. (2003). Great principles of computing. *CACM*, 46(11),15–20. Also see: <http://greatprinciples.org> with online FAQ
- Denning, P.J. (2004). The field of programmers myth, *CACM*, 47(7),15–20.
- Denning, P.J., C.H. Martell (2015). *Great Principles of Computing*. MIT Press.
- Granger, C. (2015). *Coding Is not the new literacy*. Blog post, 26 Jan. 2015. <https://www.chris-granger.com/2015/01/26/coding-is-not-the-new-literacy/>

- Heath, N. (2017). What's the future of programming? The answer lies in functional languages (an interview with Simon Peyton Jones). *TechRepublic*, 23 Oct 2017.
<https://www.techrepublic.com/article/whats-the-future-of-programming-the-answer-lies-in-functional-languages/>
- International Olympiad in Informatics*. ioinformatics.org
- IOI Syllabus* <https://ioinformatics.org/page/syllabus/12>
- Jackson, M. (1995). *Software Specifications and Requirements: A Lexicon of Practice, Principles and Prejudices*. Addison-Wesley.
- Kernighan, B.W. (2008). What should an educated person know about computers? *IEEE Solid-State Circuits Society Newsletter*, 13(2), 5–11. <https://doi.org/10.1109/N-SSC.2008.4785733>
- Kernighan, B.W. (2011). *D is for Digital: What a Well-informed Person Should Know about Computers and Communications*. CreateSpace Independent Publishing Platform.
- Kernighan, B.W. (2017). *Understanding the Digital World: What you Need to Know about Computers, the Internet, Privacy, and Security*. Princeton Univ. Press.
- Rosenbloom, P. (2013). *On Computing: The Fourth Great Scientific Domain*. The MIT Press.
- du Sautoy, M. (2019). *The Creativity Code: How AI is Learning to Write, Paint, and Think*. Fourth Estate.
- St. Amant, R. (2012). *Computing for Ordinary Mortals*. Oxford University Press.
- Tedre, M. (2014). *The Science of Computing: Shaping a Discipline*. Chapman and Hall/CRC.
- TIOBE (2019). *Programming Community Index*. <https://www.tiobe.com/tiobe-index/>
- Verhoeff, T. (2010). An Enticing Environment for Programming. *Olympiads in Informatics* 4:134–141.
- Verhoeff, T. (2011). On Abstraction and Informatics, presented at *ISSEP 2011*, Bratislava, Slovakia.
- Verhoeff, T. (2013). Informatics everywhere: Information and computation in society, science, and technology, *Olympiads in Informatics*.
<https://www.win.tue.nl/~wstomv/publications/issep-2011-on-abstraction.pdf>
errata and addenda <https://www.win.tue.nl/~wstomv/publications/abstraction-extra.pdf>
- Verhoeff, T. (2018). A master class on recursion. In: *Adventures Between Lower Bounds and Higher Altitudes* (Lecture Notes in Computer Science Vol.11011). Springer. 610–633.
DOI: https://doi.org/10.1007/978-3-319-98355-4_35
- Wikipedia Contributors (2019). Computer Go. *Wikipedia, The Free Encyclopedia*.
en.wikipedia.org/wiki/Computer_Go
Also see: www.youtube.com/watch?v=0nBpkp0FAug
- Wikipedia Contributors (2019). Deep Blue. *Wikipedia, The Free Encyclopedia*.
[en.wikipedia.org/wiki/Deep_Blue_\(chess_computer\)](https://en.wikipedia.org/wiki/Deep_Blue_(chess_computer))

A Computer Science Topics

1. Computational problems, decision problems, reduction
2. Automata (finite state, Turing machine)
3. Decidability, computability
4. Machine architecture, memory hierarchy, communication networks
5. Order analysis of algorithms, complexity (runtime, memory usage)
6. Tractability, P vs NP
7. Data encoding, compression, error detection and correction, information security
8. Reasoning about computations, formal methods
9. Data organization, databases
10. Numerical computations, floating-point arithmetic

B Programming Topics

1. Programming language, machine, operating system, interpreter, compiler
2. Imperative, object-oriented, functional, and (constraint) logic programming
3. Values, literals, types, expressions, named constants, variables, assignment

4. Input and output, formatting
5. Control flow, sequencing, conditional execution, iteration, loop invariants
6. Goal-directed algorithmic problem solving, coding idiom
7. Procedural abstraction, functions, parameters, local versus global variables, side effects
8. Modularization, data abstraction, abstract data types, recursion
9. Reuse, standard libraries, standard algorithms, idiom, design patterns
10. Event handling, (graphical) user interface, concurrency

C Software Engineering Topics

1. Coding standards, writing understandable code
2. Documentation, two-party contracts with assumptions and obligations
3. Requirements engineering, quality criteria, modeling
4. Dealing with errors, robustness, exceptions, fault tolerance
5. Decomposition (functional, data), refactoring
6. Architecture, (de)coupling, cohesion
7. Reviewing, including code review
8. Testing, unit testing, integration testing
9. Revision control, configuration management, issue tracking
10. Continuous integration, metrics, code generators, evolution, maintenance



Tom Verhoeff is Assistant Professor in Computer Science at Eindhoven University of Technology, where he works in the group Software Engineering & Technology. His research interests are support tools for verified software development and model driven engineering. He received the IOI Distinguished Service Award at IOI 2007 in Zagreb, Croatia, in particular for his role in setting up and maintaining a web archive of IOI-related material and facilities for communication in the IOI community, and in establishing, developing, chairing, and contributing to the IOI Scientific Committee from 1999 until 2007.

Computational Thinking Education through Creative Unplugged Activities

Michael WEIGEND¹, Jiří VANÍČEK², Zsuzsa PLUHÁR³, Igor PESEK⁴

¹*Holzcamp Gesamtschule Witten, University of Münster, Germany*

²*University of South Bohemia in České Budějovice, Faculty of Education, Czechia*

³*Eötvös Loránd University, Faculty of Informatics, Budapest, Hungary*

⁴*University of Maribor, Faculty of Natural Sciences and Mathematics, Slovenia*

e-mail: mw@creative-informatics.de, vanicek@pf.jcu.cz, pluharzs@caesar.elte.hu, igor.pesek@um.si

Abstract. Unplugged activities are well known in the computer science education. Creativity and computational thinking have been extensively researched and classified in last decade. In this paper we are focusing on creative unplugged activities and their potential in the classroom.

We propose a model consisting of four types of creative unplugged activities that are used in CS education and present the results of an international online survey in which 360 educators participated in 2018. The survey found out how far the model is supported by educators, the extent to which creative activities are used in the classroom, what intentions are being pursued and what educational potential is seen in the four types of activities. Based on results of the survey we present ideas and methods on how to include and integrate creative unplugged activities into CS education and some possibilities on how to change such tasks to be more creative.

Keywords: creativity, computer science, education, unplugged activities, computational thinking.

1. Introduction

A person is creative when she or he produces an idea or artefact that is new. Thus, creative persons change our culture by adding new elements. According to Czikszentmihalyi's model, creativity is not an individual ability but a system consisting of three parts:

- A *domain*, which is a system of symbols and rules (like computer science).
- A *field* that includes all persons that act as gatekeepers to the domain. They decide whether an idea or product is accepted as a new part of the domain.
- An individual person who is creative and creates new elements within the domain.

Margarete A. Boden (2007) distinguishes historical creativity (H-creativity) and psychological creativity (P-creativity). H-creativity takes place when people come up with ideas that are new and have never been shared in the history of mankind. This corresponds to Czikszenmihalyi's concept. On the other hand, P-creativity happens when a person comes up with an idea that is *subjectively new* to this person, but has been shared by someone else before.

Boden describes three types of P-creativity: combinational, exploratory, and transformational creativity. Combinational creativity involves the generation of unfamiliar (statistically unusual) combinations of familiar ideas. A typical example is a visual collage made of found photos. Visualisations of computer science (CS) concepts (using body language, Lego bricks, images) include combinational creativity. However, the CS concepts might be less familiar than the elements that are used for visualisation. Creative unplugged activities in the classroom clearly focus on combinational creativity. It is not forbidden that very bright high school students develop exploratory or even transformational creativity, but such events are beyond the scope of general education.

Czikszenmihalyi's theory is based on interviews with exceptionally creative individuals from science, art and business. For research on creativity in the classroom, we suggest considering local versions of global domains and fields.

1.1. Creativity in the CS classroom

The classroom situation can be considered as a microworld representing the society in a small scale. The parts of domains the students are familiar with are defined by the school curriculum, textbooks and learning materials on the internet. This environment can be called a "local domain". The local domain of computer science at an individual school reflects the global domain of computer science with one fundamental difference where the local domain might change rather quickly, whereas the global domain is rather stable and expands relatively slowly.

For example, with bright students and a supporting teacher in a CS class, in one year the local domain might expand quickly. Students can do unusual projects and share them with the local community. The students in the next generation might be less interested in CS, and the local CS domain shrinks again.

There is also a local field of CS at each high school teaching this subject, which mainly consists of CS teachers. They decide whether a student's work is "creative" in the meaning of "something surprising" or "unexpected". Students may also be members of the field, as they can be experts and have specific knowledge about the (local) domain of CS, and they can act as gatekeepers. The local field may be influenced by parents and students who are not in CS classes, and who are not part of the field because they are not familiar with the domain.

When creating a program, the programmer is not creative in the local domain of computer science unless she or he develops a new programming technique, which is not mentioned in the textbook or curriculum. However, she or he might have found

this »new« technique on the internet. This would also be a creative act related to the local domain if this novelty is shared and accepted by the local field. Although the idea is not new within the global domain, it is a local novelty, and the local domain is extended by this idea.

Unplugged creative tasks might be creative if the local domain is extended by fresh ideas. For example, when students visualize a recursive algorithm in a role play, they do not create a new algorithm. The novelty is the new metaphorical representations for calling a function, passing parameters, processing data and so on.

Designed activities are important elements of CS classroom education. They consist of a task and some materials which are used to solve the task. “Unplugged” activities use tangible materials like blocks or any materials that are ad hoc available, pencil and paper or just own body, but not computers. The charm of “unplugged” activities is often the contrast between the materials and the CS concepts that are elaborated.

In research literature and educational programs, one can find a lot of advice on how to encourage and foster creativity. In this section, we will try to relate these pieces of advice to creative unplugged activities, considering:

- Activity design, including the task and the used materials. Combinational creativity can be encouraged by enlarging the variety of concepts in a person’s mind. The more concepts I know, the more “unusual combinations” I can find. Additionally, a person can practice finding new associations between concepts and learn to judge the value and novelty of ideas.
- Scaffolding during the performance of the activity. An atmosphere of strength and joy should be maintained during the performance of a creative activity. The learning environment should make it easy to focus and prevent distractions. An empty school yard might be a good environment for some creative activities. Classrooms should be designed to support the creative process. Scaffolding during the process should support and encourage the following operations (Czik-szentmihalyi, 1996):
 - Clarify, analyse and re-define the problem or question to uncover new ways of looking at it.
 - Try to find connections between seemingly unrelated subject matters.
 - Challenge established wisdom by asking: how would I improve this?
 - Recognise alternative possibilities.
 - Look at things from different perspectives.
- Presentation of the results including rules for feedback. Students should not be discouraged by dismissing new and surprising ideas as mistakes. Creative people need self-confidence (Boden, 2007). The logical consequence is that encouraging a creative task should always lead to success. During the presentation, the field gets active. This is important for the creators since they must internalize the field, and they must learn to distinguish between good and bad ideas, new and old ideas. The presentation is a good opportunity to discuss novelty and values. Some ideas developed in creative unplugged tasks might get documented in some way, and they might extend the local domain of high school CS.

1.2. Computational Thinking and Creativity

In this section, we briefly describe the concept of computational thinking (CT) and its connection to creativity.

In 2017, Wing refined the definition of CT as “ Computational thinking is the thought processes involved in formulating a problem and ex-pressing its solution(s) in such a way that a computer—human or machine—can effectively carry out.”. Based on this, it can be concluded that CT involves three key components: algorithms, abstraction, and automation. Some researchers argue that it is still in an early stage of maturity with no solid definition (Lockwood and Mooney, 2017; Voogt *et al.*, 2015). In order to define and explain CT some researchers focus on CT skills (Curzon *et al.*, 2014; Atmatzidou and Demetriadis, 2016), others on its elements (Grover and Pea, 2013) where one of the elements is abstraction and pattern generalization (including modeling and simulation). Some researchers argue that CT is an activity, often associated with, but not limited to, problem solving (CSTA and ISTE, 2011, Beecher, 2017, p. 8, Haseski *et al.*, 2018).

CT can be seen as a fundamental skill for everyone, not just for computer scientists. It is applicable in either a computerised or unplugged problem-solving process. CT has the potential for application in a wide range of disciplines as the creative learning arrangements. Two main strategies are used for CT skills development: unplugged activities (activities that involve logic games, cards, puzzles, strings or physical movements to get in touch with computer science concepts such as algorithms, data transmission or data representation) and computerized activities (such as programming in arrow-based visual environments, programming in block-based visual environments, using textual programming languages, programming that is connected with the physical world) (Moreno-León *et al.*, 2018).

In this paper, we look only on creative aspect of CT. According to Korkmaz *et al.* (2017), CT is the extension of the problem solving skills of a person and the development of the creativity and critical thinking skills of the people by re-focusing. Thus creativity plays an important role in CT approach (Korkmaz, 2017).

CT fosters creativity in the classroom as it allows students to move from being technology consumers to technology developer. Creativity can also be augmented by CT (Mishra *et al.*, 2013).

In the field of CS, creativity usually aims at producing new techniques, both hardware and software, that can provide solutions to practical problems. However, creativity involves the same kinds of cognitive processes that generate answers in computing as well as in the natural sciences (Saunders and Thagard, 2005). Moreover, in CT, creativity could be seen as an ability of applying imagination to create a physical object or some mental or emotional construct (Korkmaz *et al.*, 2017) that is judged to be novel and also to be appropriate, useful, or valuable by a suitably knowledgeable social group (DeSchryver and Yadav, 2015).

1.3. *Unplugged Computer Science Activities in the Classroom*

Many important topics in informatics or in other areas of science, society and technology connected with computer science can be taught without using computers.

The CS unplugged activities provide a scaffolding for a constructivist approach to introducing topics in computer science, without the need to learn programming first (Bell, 2018). The connections and some ideas and working processes are often easier to explain with hand-made activities (Bell *et al.*, 2012). These activities can be simpler and can be performed without computer. They can be used to present the important parts (e.g. sub-knowledge) of a big system or a working process in a concept.

Unplugged CS activities support computational thinking, although for this to be effective they should be used in a context where they will be linked to implementation on a digital device (Bell, 2018). There is another important aspect: by using these tasks, they could help to “de-mythize” the computer in the eyes of children as a “magic box who knows and does a lot and we do not understand how”.

In this section, we briefly summarize the advantages and disadvantages of using unplugged activities in the classroom.

1.4. *Advantages of Using Unplugged Activities in the Classroom*

Using unplugged activities helps to think not only about computers and computer science, but about interdisciplinary activities without school-subject boundaries. Students can see the relationship(s) between different subjects and disciplines better.

The activities are mostly combined with physical movements. The aims of theories about learning by doing and learning by moving promote breaking out of the daily routines and school-life. (Bell *et al.*, 2015; Rodriguez *et al.*, 2017; Thies and Wahrenhold, 2013)

The activities give the basis for cooperation and teamwork. They can support the development of communication as skill. Students need to speak about a problem: they need to formulate the problem, the solving strategy, and the solution. (Bell *et al.*, 2015; Feaster *et al.*, 2011) In some activities, they need to present artifacts to an audience and handle the comments.

1.5. *Disadvantages (Problems, Challenges) of Using Unplugged Activities*

The computer unplugged activities are not only about “playing”. They need a strong basis: a stable preparation. Mostly they give a lot of work for teachers. Not only in preparation (initiative) processes, but they can be barrier for the teacher. They need to change their mind from “exact” computer science to non-exact open ended situations; which necessitates changing their teaching style. It is hard work to keep the focus (why we started) and to manage time, as well.

Students can refuse to do the activity – “playing” is not always a means of motivation. (Rodriguez *et al.*, 2017; Thies and Wahrenhold, 2013). We present some additional comments about the problems of using unplugged activities in chapter Research.

2. Definitions of Creative Unplugged Tasks

In this section, we present a model for categorizing different types of creative unplugged classroom activities on computational thinking. The model consists of four categories which are described and presented with an example each.

Category 1: Invent an algorithm

Invent an algorithm that solves a given task and present it without a computer.

Here, the creative challenge lies in devising and formulating an algorithm with appropriate commands and data representations.

Example: Invent with your team how to move a sequence of 0s and 1s in the playground from A to B across multiple stations. Only body language may be used.

Possible solution: Represent a zero by stretching out both hands and represent a one by a jump.

Category 2: Find an application

For a computer science algorithm or concept, find a new situation in which this algorithm or concept could be applied.

Here, the creative challenge lies in the design of a story. The algorithm is given. It needs to be understood so that you can check if the story fits as the application context.

Example: Describe a situation in which you can use the binary search. Show the situation in a small role play. Use things from this room.

Possible solution: Search for a word in a dictionary.

Category 3: Find an example

Find an algorithm that has certain given structural features (e.g., loops, recursion, function calls) and represent it in some way.

Here the creative challenge lies in finding both a task and a suitable algorithm. These kinds of creative tasks solve teachers e.g. when looking for illustrative examples of a specific programming technique.

Example: Here is the definition of a function with a parameter in Python notation.

```
def crumble (thing):
    Open the hand.
    Put the matter in the hand.
    while not thing has the form of a ball:
        Move the fingers of the hand.
        Give it back.

# Main program
crumble(sheet of paper)
```

Imagine a program in the style of the example and write it down. The program text should contain a function definition with at least two parameters. As parameters you use everyday objects from this room.

Possible solution: Wrap a thing in a foil.

Category 4: Find a visualization

Invent a visualization for an algorithm or concept of computer science.

Here the creative challenge lies in finding a representation that is understandable to the public. Various materials may be used, e.g. crayons and paper; role-playing with props, pantomime, Lego building blocks. These kinds of creative tasks solve e.g. book authors when looking for illustrations for a textbook.

Example: There are cards with terms from computer science, which should be visualized. Pupils draw a card and present the term through mime. The audience has to find out what it is.

Possible solution: Visualization of a WLAN access point through mime.

3. Research

Survey

In 2018, an international online survey was conducted. The questionnaire was translated to the national languages, and it was sent through different channels to computer science educators. In Table 1, the distribution of the educators based on their country of origin is presented, with the total number of answers, 360.

The respondents’ pedagogical expertise is presented in Table 2. Most of them have a background in secondary education.

In the questionnaire, we first presented four different types of creative unplugged activities, and then asked the respondents how clear they thought the definitions were. The responses were given on a scale from unclear (1) to clear (5). Fig. 1 presents the results.

107 out of the 360 respondents (more than ¼) claimed that they know other creative CS activities without computer besides our four categories. 38 respondents gave one or more examples, some respondents wrote general comments.

Table 1
Respondents origin

Country of origin	Number of respondents
Czech republic	133
Lithuania	8
Slovenia	25
Germany	153
Japan	14
Other countries	27
Total	360

Table 2
Respondents' professional experience

Pedagogical expertise	Percentage
Teaching computer science-related content at a primary school	28 %
Teaching computer science related content at a secondary school	88 %
Teaching computer science related content at a university	14 %
Authoring or co-authoring a textbook in the field of computer science	12 %
Research in the field of computer science	13 %
Teacher training in the field of computer science	27 %

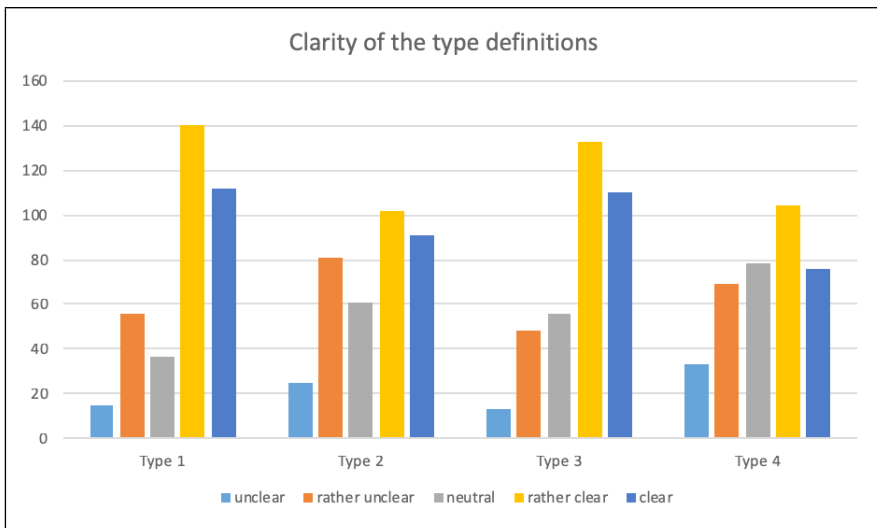


Fig. 1. Graph representing clarity of the categorization.

We can identify only about one third of these 38 responses as examples of creative unplugged activities, moreover, most of the presented examples were unplugged tasks but were not creative, as they were typical problem-solving activities (e.g. ordering data, comparison of algorithms). Some examples were not even from field of CS (e.g. painting). In some tasks, it was difficult to decide whether this task is creative or not, for example, the response „puzzles – sudoku“ we could not decide whether the respondent meant solving sudoku or creating a new puzzle.

Some of the responses described concrete creative activities which were written very sketchy, yet we can say that they can be categorized to one of our four types e.g. „Representing the TCP/IP protocol stack through role play using envelopes as packets“ which is Type 4, create a visualization.

We can say that only a few answers out of the 38 responses were creative unplugged activities that cannot be directly mapped to our model of activity types. These answers were not oriented to algorithms but to data transfer, data coding or finding a pattern. If

we wanted to create one new type of activities based on the questionnaire responses, we could find it probably in „more complex tasks“, „systematisation“, „discussions about possible solving strategies“ or „demand of informatics in the society, economy“.

Next we asked the respondents about their experiences with creative tasks. On the question “How often have you spent the last 12 months doing creative tasks without a computer in computer science education or a computer science course?”, 50% of the respondents said they used them once or twice, 27% used it more than twice and 23% said they never done them in the classroom.

On the question “If you have engaged in creative tasks what was the reaction of the students?” we got the results presented in Table 3.

59,4% of all participants observed students’ positive reactions and 10.7% of all participants negative or no reactions.

Next we asked educators if they could tell us why and how they used creative unplugged activities. Respondents could check multiple options. The results are presented in Table 4.

We asked the educators about their opinion on the educational potential of creative tasks. More precisely, we asked them about specific potential and which type of creative tasks can be used instead. More answers could be selected at each question. Results for different purposes are in Table 5.

Research on the results of the educators with the positive students’ feedback

As the next step, we compared the attitudes of educators who reported positive and rather positive students’ reactions with those who got a non-positive reaction (neutral,

Table 3
Students reaction

Student’s reaction	Percentage
Positive	13.6 %
Rather positive	45.8 %
Neutral	12.7 %
Rather negative	6.6 %
Negative	0.5 %
No reaction	3.6 %
No answer	16.9 %

Table 4
Usage of creative unplugged activities by teachers

Intention	Percentage
As an activating entry into a new topic	51 %
As a suggestion to think about certain abstract concepts	47 %
To promote creativity	26 %
As training, e.g. to practice the correct use of technical terms	25 %
As a diagnostic tool to identify a prior understanding of a topic or misconceptions	16 %
As an entertainment element to relax a course	38 %

Table 5
Educational potential of 4 types of CS tasks.

Educational Potential	Type 1 <i>Create an algorithm</i>	Type 2 <i>Find an application</i>	Type 3 <i>Find an example</i>	Type 4 <i>Visualisa- tion</i>
A replacement for programming tasks providing a comparable learning experience in less time	72 %	50 %	44 %	46 %
An enrichment to a lesson or lecture to make the content more relevant and attractive to the participants	68 %	58 %	39 %	62 %
As a encouragement for students to think about computer science concepts	61 %	67 %	54 %	60 %
As a help to develop transferable skills	62 %	71 %	49 %	56 %

Table 6
Attitudes of educators with different reaction

Intention	Positive reaction group (N = 214)	Non-positive reaction group (N = 85)	Chi-Square
As activating entry	62.1 %	57.6 %	.472
As a suggestion to think about certain abstract concepts	61.2 %	42.4 %	.003
To promote creativity	20.0 %	35.5 %	.009
As training, e.g. to practice the correct use of technical terms	30.6 %	29.4 %	.845
As a diagnostic tool to identify a prior understanding of a topic or misconceptions	15.3 %	20.6 %	.296
As an entertainment element to relax a course	36.5 %	48.1 %	.067

rather negative, negative). In the first group we have 214 respondents and in the second group, 85 respondents. We excluded the respondents (N = 64) who did not provide an answer.

We used Pearson's Chi-square test to check if there exists a statistically significant association between positive reactions of the group, and the intentions of usage in the classroom. Results are in Table 6.

We also checked whether there exist any differences in their opinions regarding the educational potential of the four types of unplugged creative activities. The results show that both groups share similar views ($p > 0.5$), except when activity is viewed as a encouragement for students to think about computer science concepts where $p = 0.05$ for type 3 and $p = 0.03$ for type 4.

Discussion

From Fig. 1 it can be observed that types 1 and 3 are clearer and better understandable for the educators than type 2 and 4. It's interesting because type 2 and type 3 are very similar. It seems that educators feel more confident with activities that ask for specific structural

features than with more general concepts. One of the reasons is that activities and tasks that fit in type 3 category are more frequently used in CS teaching.

We can conclude from the answers presented in Table 4 that educators use creative unplugged activities as activating entries or as suggestions to think about some concepts, which are also the same reasons why they are used to promote unplugged activities. It's interesting to notice that more than one third of educators use unplugged activities also as an entertainment element in the classrooms.

As a replacement for programming tasks presented in Table 5 most educators think that activities that fit into Type 1 (Create an algorithm) are most useful and appropriate. The reason for such results might lie in the fact that Type 1 activities are conceptually very similar to the programming tasks, only the allowed tools are different.

For the other three types, it's not that easy to see how such activities can replace programming tasks. It's interesting to notice that educators don't feel that Type 3 activities can replace programming tasks, although such activities develop understanding of some concepts similar to programming tasks. Most of educators think that all types of creative tasks, except Type 3 tasks, are suitable for use in the classrooms as enrichment.

Analysing the answers in depth, we can see that almost 90% of respondents that checked Type 3 also checked tasks of Type 1 and Type 2. Educators think that tasks of Type 2 are most appropriate to be used as encouragement, although other types of activities are also used in classrooms. We can conclude that educators think that finding a new situation encourages students to think about CS and also helps students to use their CS knowledge in different areas of their life. Most educators think that tasks of Type 1 and 2 help students develop skills that can be used in other scientific fields. The reason might lie in the fact that educators think that the first two types help in developing computational thinking which can be later applied in various fields.

Educators who reported positive students' reactions on creative unplugged activities ("successful educators") tended to use these activities more frequently to encourage thinking about CS concepts and *less* to encourage creativity in general. This could be a hint that these educators were successful *because* they tried to encourage thinking about CS concepts. However the perception of students' reactions may be subjective.

The fact that almost 60% educators noticed positive reaction must be used to further encourage the use of unplugged activities in classrooms.

The questionnaire explained four types of creative unplugged tasks and gave examples. For many CS teachers, this might have been the first contact with unplugged activities and creativity in the classroom, and they might have had problems with understanding short introduction of the survey.

A possible reason that teachers do not use creative unplugged tasks in CS education so much might be that focusing on computational thinking is not established so well in CS education, and the fact that there aren't that many available prepared creative unplugged activities which educators could use. Another reason could be the perceived lack of time for these activities which seem to be time-consuming.

From curricula point of view, teachers saw application of these tasks as an introduction to understanding how computers work, in media education and painting art activi-

ties and in creating criteria by ordering data. This might indicate that some teachers connect creativity more to using ICT than to computer science.

The results of our survey show that the creative unplugged activities are not used often in education. Therefore, in next chapter we propose suggestions how to prepare and use creative unplugged activities to develop and encourage creativity in computer science education.

4. Suggestions for Creative Unplugged Activities in Computers Science Education

In this section, we present some ideas how to develop creative CS unplugged classroom activities fulfilling three conditions:

- They challenge creativity.
- They touch a computer science topic.
- They can be performed without a computer.

The first three subsections discuss how to create a new creative CS task, subsection 4.4. shows how to use the created activities in the learning process.

4.1. *How to Express Ideas*

In this section we focus on the potential and challenges of different “unplugged” expressive means: writing, drawing, concept cartoon, building with physical constructions bricks or found material and role plays. Generally, each type of creative activity can be performed using different expressive means.

Writing

The most obvious way to express an idea is just to write it down. The product could be just a list of words or a story with an action plot.

Example tasks:

- Write down the idea of an exciting programming project that is based on case distinctions, which could be coded by if-elif-else statements.
- Write a christmas story, in which an iteration takes place.

Drawing

Students may create drawings alone or collaboratively in a team. The advantage of collaborative drawing is that there is more communication. Two or more students sit at a table around a big sheet of paper. They can partly work independently. Each person might be responsible for a certain part of the image. Example tasks:

- Think of five or six substances you can find in this room (iron, wood, ...). Create an image that illustrates how to distinguish these substances (decision tree). (Type 1)

- Draw a robot ship with sensors that can find and remove plastic garbage from the ground of the ocean. (Type 1)
- Many computer programs contain while-statements. In a while-statement a sequence of activities is executed again and again as long as a certain condition is true. Example: “While you are thirsty, put some water in a glass and drink it.” In this case, the condition is “you are thirsty”, and the sequence of activities is “put some water in a glass and drink it”. Create a comic-book story that illustrates a while-statement taking place in everyday life or in a fantasy world. (Type 3)

There are free web-based systems that support collaborative drawing, so called “web whiteboards” (e.g. Ziteboard, LiveBoard, Explain Everything Whiteboard). Students may use their smartphones or tablets to create a collaborative visualization or algorithmic idea with this technology.

Concept cartoons

Concept cartoons (Naylor and Keogh, 1993) are comic-like illustrations depicting dialogs or thoughts of individual persons. The challenge is to explain a scientific issue using everyday language. Although they mainly consist of written text, concept cartoons are visualisations (type 4). They serve to express ideas on a topic from different points of view thus encouraging diversity of thinking. Concept cartoons in textbooks sometimes explicate typical misconceptions, that provoke discussion and falsification.

A concept cartoon-related task may be completely open. Example:

“Tina visualizes a variable name by a sticky note that is put at an object. For example, the Python statement `a = "car"` she represents by the string literal “car” carrying a sticky note with the name “a” written on it. Draw a cartoon depicting Tina explaining this program.”

A very common and less open version of concept cartoon-tasks provides drawings, empty speech bubbles and a start. The challenge is to continue the story. Figure X shows an example.

Finalized concept cartoons can later be used as as basis for discussions for instance on the limitations of metaphors that were used in the dialogs.

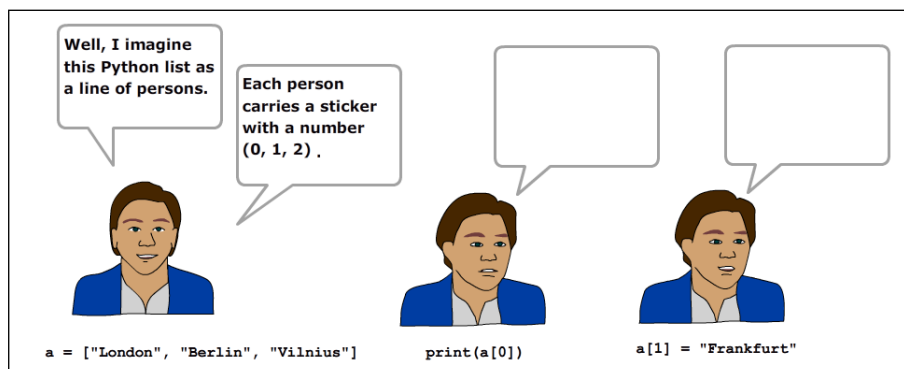


Fig. 2. Concept cartoon.

Construction bricks

“Think with your hands” is the motto of representing abstract ideas with Lego. A participant picks a concept or question (like “How does a digital camera take a picture?”) and creates (within a few minutes) a sculpture visualizing the idea (type 4) using all kinds of LEGO blocks. Then other participants have to find out what it is. Finally the creator explains his or her artifact. This activity should be done in small groups of four people sitting around a table. It includes a lot of communication and usually new and surprising ideas are exchanged.

There are two important rules for LEGO sculpturing that should be clarified first: 1) If you do not know what to build, just start building. The ideas will come later. 2) Everything, what you build is just fine.

Building with construction bricks requires a “warming up” to get in the mood of “thinking with your hands”. A well-known procedure is the LEGO duck. Each participant gets six bricks and creates a duck in 60 seconds. There are many ways to solve the task. Even if you do nothing you have solved the task. In this case, the bricks are just the parts of a roasted and carved duck.

Instead of LEGO one can also use found material, things like pencil, ruler, paper sheet that are always present in a classroom. However, LEGO offers a greater diversity of material and the created artifact always looks nice and interesting.

Theatre play

People are used to role playing since childhood. Playing different roles is important for the development of social skills like empathy and tolerance of ambiguity. In CS education improvised role plays can be used to visualise algorithms or to give.

The many ways to organize role plays can be grouped in two types: Role plays that are prepared and performed independently by small groups and role plays in the style of “interactive theatre” that involve the whole audience and are controlled by the teacher or lecturer.

Role play in small groups could be a classic role play with dialogs and props, a mime or a “living statue”. Here are examples for tasks of different types.

- **Type 1:** Create an algorithm for building a tower with LEGO bricks with four persons as quickly as possible.
- **Type 2:** Play a situation in which redundancy is important. Start with deciding where the story should take place (on the school yard, at a river, on the moon, ...).
- **Type 3:** Play the execution of a function with two parameters. The parameters should represent things from this room. The function should return something.
- **Type 4:** Visualize the “Bubble Sort” algorithm in a silent theatre play.

An algorithm can be visualised or developed in the style of interactive theatre. Audience participation is quite established in theatre for children (Way, 1981). Some ideas of this tradition can be used in CS education. The lecturer encourages the audience to talk and play but still controls the development of the plot.

- **Type 1:** The lecturer develops an algorithm together with the audience. The audience tries out each idea immediately, discusses the solution and improves it, getting into an iterative process of constantly rethinking, adjusting and redoing.

Gerald Futschek (University of Vienna) is well known for this approach (see for example Futschek and Moschitz, 2010).

- **Type 4:** The lecturer directs a role play visualizing an algorithm that is difficult to understand, like a recursive version of the Quicksort sorting algorithm (Hoare). The props of the play are the algorithm, for example represented by a Python function definition, and a stack of big DUPLO bricks of different sizes and colors. Members of the audience play processors that can execute a function call according to the given code. The lecturer starts the play by passing the stack of bricks to one member of the audience. She or he scaffolds the play – and keeps the plot on track – by asking questions and initializing dialogues instead of explaining. Talking is essential for connecting the written code to the action of the play.

4.2. How to Add Creativity to CS Tasks

In this section we focus on potential of tasks from the Bebras challenge contest (Dagienė, 2017) as a good example of CS tasks which are not open ended. In these tasks, students plunge into a described situation which they must grasp, get to understand the concepts and terms that are used, find an informatics principle the task is based on and solve the problem using cognitive and thinking skills. These tasks are problem tasks, not creative. We can find several ways of involving creativity when solving Bebras task:

- Change the type of answer from multiple-choice to interactive.
- Move the Bebras tasks away from the computer.
- Take the Bebras tasks as inspiration for new tasks.

Change the Type of Answer of Bebras Task from Multiple-Choice to Interactive

Interactive Bebras tasks mean tasks where contestants give the answer by dragging objects on the screen or click on them. This allows to use tasks with more than one correct answer (and require to find at least one or all correct solutions).

Compare two versions of the same Bebras task about discovering Euler condition (title Even connections, 2017-PL-04):

A company has to connect 10 access points with data cables (lines) in one net according to a plan. Unfortunately this company has no technology to cut the cable to pieces so that it still worked. They can create one cable loop going through all access points and visit one point more than once.

Now it is impossible to solve this problem and it is necessary to remove some lines in the plan.

(multiple-choice version – Fig. 3 left): Which one of lines A, B, C, D is a part of the best solution of the problem if we want to remove as few lines as possible so that access points were connected without cutting the cable? Choose one of A, B, C, D. (Correct solution is C because it allows to remove only 3 lines in the graph; removing A, B or D lines requires to remove more than 3 lines in the graph).

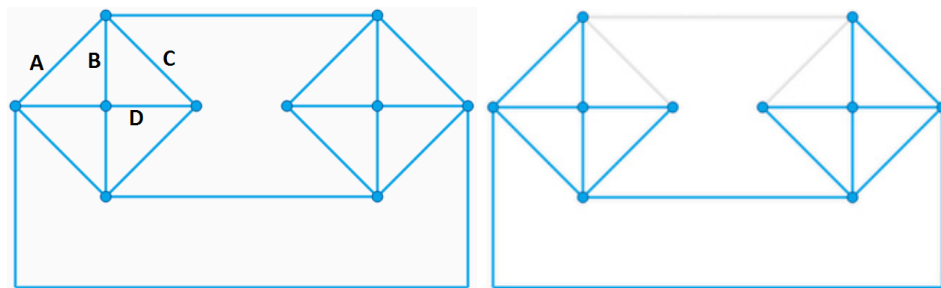


Fig. 3. Multiple-choice and interactive version of a Bebras task.

(interactive version – Fig. 3 right): Remove some of the lines by clicking on them so that the problem is solved by removing as few lines as possible. There is more than one correct solution.

Move Bebras Tasks Away From the Computer

Most of the Bebras tasks are ready to use without a computer. They have a story with objects as basics. We can bring these stories to life, and we can use these objects in our real world.

Instead of clicking, or dragging and dropping on the screen, we can move the objects, construct and “build” new buildings, sequences or objects. The success of problem solving strategies can be tested, and the teacher has the opportunity to follow along the strategies used, to help building and learning new strategies.

A typical example can be the task entitled Loading Lisa’s (2014-DE-08), where we need to put barrels onto the boats so that each boat gets loaded with as many kilograms of water as possible. On the screen, we can make an interactive drag and drop game. As an activity without computer, we might have more preparation ideas: using a magnet wand with magnet object; or we can prepare weight-objects (sacks with sands or cans with water with different weights), and two boats made of carton boxes. Students can move the weight objects, and experience with the solution – find the best strategy to solve the problem.

But not only drag-and-drop exercises can be converted.

- Students could become more creative when they can use their hands: building a construction from small parts – or the reverse version: find the small repeatable part in the big one.
- The real world can help solving 3D or other visualisation, such as origami, repeated buildings or activities with a paper or 3 dimensional objects.

In a challenge game or in an escape room, we can connect these activities: the first activity can give the instruction to the others, or more activity-results can add together the whole solution (or the path to the solution). Throughout these activities, participants can learn (or use) scheduling and sharing skills: they have the opportunity to work on tasks together. However, there are some problems that are best to be solved simultaneously, so participants need to define roles, and divide the activities into sub-activities.

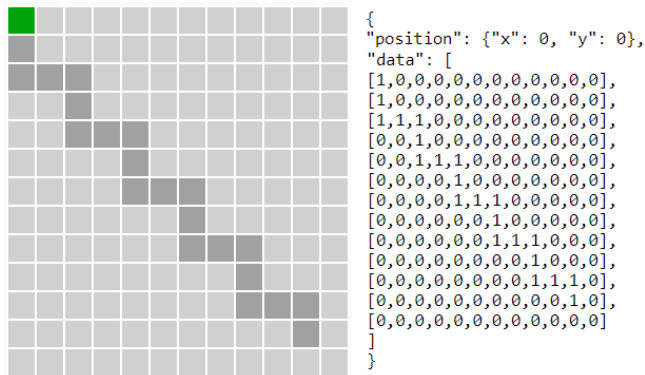


Fig. 4. Interactive Bebras task – situation visible for contestants and initial data.

Taking Bebras Tasks as Inspiration for New Tasks

Bebras tasks could be a source of inspiration for changing something to create a new task. The new task is of course not creative but students are creative during its creation, they become “tasks creators” for other students, for competition participants.

- We can let students change a story or picture so that the problem would give a new look.
- Students can change part of the settings (e.g. initial word, number or string) of the task so that the kind of thinking will stay but the situation will change a bit.
- If an interactive task is prepared, the user can change the settings easily, the students can change the initial situation so that a new problem different from the original task occurs.

One example of such task is visible in Fig. 4: the interactive Bebras task Painting robot (2013-FR-05). The user has to write a program for the robot (green point in the corner) so that it goes through all dark fields in the playing field. The robot understands commands of cardinal directions movement and simple loops. Students can change initial data of this task (see Fig. 4 on the right) changing required shape to mark with possibility to create more complex or clever task than the initial one.

Do not expect that students as task creators will always be creative. Often they create something like variation of original task by making non-important changes in the settings, sometimes only by changing a story. But sometimes they could work with a completely new idea: more complex shape, nested commands, hidden connections between objects to be discovered...

4.3. Adding CS to Creative Tasks

It is well known that many user approach activities with ICT could be good starting points to inquiry, to discover computer science rules, to move from application to the core topic of computer science. E.g. painting in computer could move to thinking about

pixels, colour depth, graphic formats. Writing texts could move to coding, data transfer, data compression. Work in vector graphics could result in logic operations and algorithms how to paint something. Creative work with digital technology can result in creative work in computer science, sometimes with unplugged activities.

We can develop creativity using problems with multiple solutions. Our experience show that it is hard for teachers to promote creativity in a quite new topic for students which unplugged computer science really is. One method is to show several ways of solving some open problems. Students, then, can imagine that such tasks allow different approaches of solving it, and might be more willing to create another.

Let us illustrate this approach on activity about how to code the data transferred. When electricity is supplied from external battery, Micro:bit can work unplugged without computer and can be used the user way when some program is downloaded to it. This activity uses Micro:bit as a tool for creating icons (picture).

Students use prepared program in which a cursor goes through all pixels in a matrix. They create their own icon by buttons A (light up), B (turn off) on the board (Fig. 5). Then students describe their icon by letters A, B, in the order they have touched these buttons, and give the code to the other student to create the icon in her/his device. In this part of activity, students are creative at user level. The role of Micro:bit is to check the rules (and of course to make the activity more attractive), this activity could be realized without this device.

In the second step, students create an icon using different given methods, as a path of movement of Karel the robot (a blinking point on the grid) controlled by two commands: forward (button A) and right turn (button B). Students could get that there are different methods of describing the icon using the codes of A's and B's, and might realize that without telling the method of coding it is hard to reproduce original work.

Finally students have to invent a “their own”, new secret code which can describe the icon so that only the receiver could reproduce it. Students can give different meanings to letters A and B (e.g. A sets painting pen up or down, and B moves the pen on the grid), or go through the grid in a different way (from the right, by columns ...). This creative activity could result in classroom discussion whether new methods of coding are correct, unambiguous and suitable for all icons. Students are creative as computer scientists in this part of the activity.

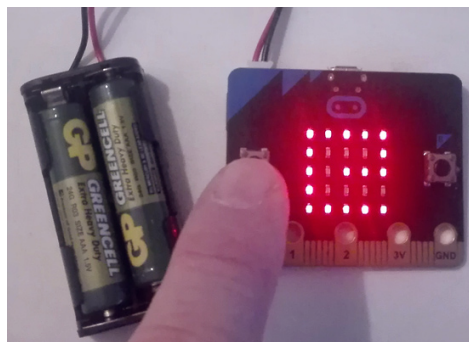


Fig. 5. Coding icons in Micro:bit.

4.4. How to Embed Creative Unplugged Activities in the Learning Processes

Creative artifacts are a contribution to the knowledge and culture of the learning community. Students should be aware of this value. In this section, we discuss a few ways of presenting the outcomes of creative activities, and how to embed them in the learning process of a class or lecture. On the web, artefacts can easily be published worldwide. However, according to the idea of the “local domain” of a learning community (see above) we focus on the presentation within the protected space of an educational institution.

Puzzle and Quiz

Examples of CS concepts, which students have created (type 2), may be used for a puzzle or quiz-like classroom activity. For example, after finding examples for list structures (Table 7) participants write Python list literals at the blackboard and for each literal the audience has to find out which real object in the room is modelled by it.

Similar puzzles could involve Lego artefacts or photos visualizing a concept (type 4), or stories that are related to an algorithm or concept (type 2).

A puzzle is an explicit cognitive challenge to the audience. Besides solving the puzzle, the audience is also encouraged to validate solutions, and to evaluate the quality of the visualization.

Exhibition

Artefacts that have some aesthetic value (drawings, sculptures, photos) can be collected and exhibited at a classroom wall or in a glass case. Example: One spot at the wall exhibits photos of all kinds of sorted sequences (jackets on hangers in a shop, floors of a building, books in the library, deck of cards). An exhibition – created by students – is not just a nice decoration but it reflects the students’ everyday life, what is relevant to them and the way they think and feel.

The exhibits can later be used for programming exercises. For example, the “collection of sorted things” hanging at the wall might be a reason to think about how to model these entities using formal code. A deck of cards could be represented by a list of 52 tuples of the form (suit, card).

Performance

Roll play and mime performances are not easy to record. There are two major problems: It takes much time, competence and effort to record a performance and to create a good

Table 7
Finding examples for list structures

Python list literal (model)	Object in the classroom (reality)
<code>[[3, 4, 4], [3, 4, 2]]</code>	Two rows of tables with different numbers of students sitting at them
<code>[7, 7]</code>	Illumination at the ceiling consisting of two rows with seven lamps each
<code>[[0111, 1111, 1111], [1011, 1111, 0011]]</code>	Two rows of tables with four chairs each. The digits 1 and 0 indicate whether or not somebody is sitting on this chair

video. One has to think of camera perspectives, frames, sound and editing. You can find many videos documenting role plays on Youtube which are not only boring to watch but which are also difficult to understand.

The second problem is the personal rights of the actors. A video may be embarrassing for the actors, and it can only be published legally if the actors (and eventually their parents) have signed an agreement. So, in most cases, performances are live and remain undocumented. However, it is possible to take photos of scenes in way that the faces of the actors cannot be recognized.

If a live performance is dynamic (like an improvised role play), it may represent the idea of a new algorithm or a use case in an easily comprehensible way (type 1, 2 or 3). Such performance is done quickly, and it could be the starting point of a programming project. Example: Play sorting a deck of cards using straight selection and then write a sort function.

5. Conclusion

The spectrum of creative unplugged activities on CS can be grouped into four types: 1) Create an algorithm 2) Find an application 3) Find an example 4) Find a visualisation. This classification seems to be sufficient for most CS educators that have been asked in a questionnaire.

CS educators use creative unplugged activities for several reasons, especially as activating introduction into a new topic and to encourage students to think about CS concepts. Generally they see educational potential in all types. However, most educators rarely use these activities (a few times a year).

This may indicate that the repertoire of useful creative unplugged activities for CS education is still small and needs to be extended by the community of professional educators. Teachers developing new ideas for creative classroom activities might consider these points:

- There are different types of activities, which imply different ways of how to get creative. Being creative is joyful. Offering a diversity of tasks makes it easier for the individual student to find an activity to which she or he is able to contribute.
- There are many ways to express ideas (writing, drawing, building, playing, ...). A student may pick a method she or he is good at to create.
- The result of a creative process is a new product (a new story, a new picture, a new sculpture, a new play etc.) that extends the “local domain” of relevant CS knowledge within a certain class at a certain school. This product can be presented and exhibited and is valuable for learning.
- There is a repertoire of successful creative activities in school education. Well known creative activities can be adopted to CS topics.
- There are many well thought – but not creative – unplugged activities in CS education including paper versions of Bebras tasks and exercises from the book “CS unplugged”. Developers of educational material could take this wealth of ideas as inspiration and make closed and analytical tasks more creative.

References

- Atmatzidou, S., Demetriadis, S. (2016). Advancing students' computational thinking skills through educational robotics: A study on age and gender relevant differences. *Robotics and Autonomous Systems*, 75, 661–670.
- Beecher, K. (2017). *Computational Thinking: A Beginner's Guide to Problem-Solving and Programming*. Swindon BCS.
- Bell, T. (2018). CS Unplugged and Computational thinking, In: *Proceedings of Constructionism 2018, Vilnius, Lithuania*, p. 22 – 28.
- Bell, T., Rosamond, F., Casey, N. (2012). Computer Science Unplugged and related projects in math and computer science popularization. In: Bodlaender H.L., Downey R., Fomin F.V., Marx D. (Eds.), *The Multivariate Algorithmic Revolution and Beyond*. Lecture Notes in Computer Science, vol 7370. Springer, Berlin, Heidelberg
- Bell, T., Witten, I.H., Fellows, M. (2015). *CS Unplugged*. www.csunplugged.org
- Boden, M. (2007). How creativity works, *Creativity East Midlands for the Creativity: Innovation and Industry conference*, 2007.
- Rodriguez, B., Kennicutt, S., Rader, C., Camp, T. (2017). Assessing computational thinking in CS unplugged activities. In: *Proceeding SIGCSE '17 Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, p. 501–506.
- Csikszentmihalyi, M. (1996). *Creativity: Flow and the Psychology of Discovery and Invention*, Harper Collins Publishers, New York
- Curzon, P., McOwan, P.W., Plant, N., Meagher, L.R. (2014). Introducing teachers to computational thinking using unplugged storytelling. In: *ACM Proceedings of the 9th workshop in primary and secondary computing education*. 89–92.
- Dagienė, V., Sentance, S., Stupurienė, G. (2017). Developing a two-dimensional categorization system for educational tasks in informatics. *Informatica*, 28(1), 23–44.
- DeSchryver, M.D., Yadav, A. (2015). Creative and computational thinking in the context of new literacies: Working with teachers to scaffold complex technology-mediated approaches to teaching and learning. *Journal of Technology and Teacher Education*, 23(3), 411–431.
- Feaster, Y., Segars, L., Wahba, S., O. Hallstrom, J. (2011). Teaching CS unplugged in the high school (with limited success). IN: *ITiCSE'11 – Proceedings of the 16th Annual Conference on Innovation and Technology in Computer Science*. 248–252.
- Futschek, G., Moschitz, J. (2010). Developing algorithmic thinking by inventing and playing algorithms. In: Clayson, J.E., Kala, I. (Eds.) *Constructionist Approaches to Creative Learning, Thinking and Education: Lessons for the 21st Century, Proceedings Constructionism 2010*, Paris 16.-20. 8. 2010.
- Grover, S., Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational Researcher*, 42(1), 38–43.
- Haseski, H.I., Ilic, U., Tugtekin, U. (2018). Defining a new 21st century skill-computational thinking: Concepts and trends. *International Education Studies*, 11(4), 29.
- ISTE, CSTA. (2011). *Computational Thinking in K–12 Education leadership toolkit*.
- Korkmaz, Ö., Çakir, R., Özden, M.Y. (2017). A validity and reliability study of the Computational Thinking Scales (CTS). *Computers in Human Behavior*, 72, 558–569.
- Lockwood, J., Mooney, A. (2017). Computational Thinking in Education: Where does it fit? A systematic literary review. arXiv preprint. *arXiv:1703.07659*.
- Mishra, P., Yadav, A., Deep-Play Research Group. (2013). Rethinking technology & creativity in the 21st century. *TechTrends*, 57(3), 10–14.
- Moreno-León, J., Román-González, M., Robles, G. (2018). On computational thinking as a universal skill: A review of the latest research on this ability. In: *Global Engineering Education Conference (EDUCON) 2018*, IEEE. 1684–1689.
- Saunders, D., Thagard, P. (2005). Creativity in computer science. *Creativity Across Domains: Faces of the Muse*, 153–167.
- Thies, R., Vahrenhold, J. (2013). On plugging “unplugged” into CS classes. In: *Special Interest Group on Computer Science Education*, Denver, 2013.
- Voogt, J., Fisser, P., Good, J., Mishra, P., Yadav, A. (2015). Computational thinking in compulsory education: Towards an agenda for research and practice. *Education and Information Technologies*, 20(4), 715–728.

- Weigend, M., Pluhar, Z., Juškevičienė, A., Vaníček, J., Ito, K., Pesek, I. (2018). Constructionism in the Classroom: Creative Learning Activities on Computational Thinking. *Constructionism 2018 Conference Proceedings*. URL: http://www.constructionism2018.fsf.vu.lt/file/repository/Proceeding_2018_Constructionism.pdf
- Wing, J.M. (2017). Computational thinking's influence on research and education for all. *Italian Journal of Educational Technology*, 25(2), 7–14. DOI: 10.17471/2499-4324/922



M. Weigend studied Chemistry, Pedagogy and Computer Science at the University of Bochum and the University of Hagen and received a PhD in Computer Science from the University of Potsdam. He is a teacher at a secondary school in Witten, Germany, lectures at the University of Münster and he has taught Didactics of Computer Science at the University of Hagen for almost 20 years. He has published several books on computer programming and visual modeling.



J. Vaníček is associated professor and head of the Department of Informatics at the Faculty of Education, University of South Bohemia in České Budějovice, Czech Republic. He prepares primary and CS teachers and takes care of informatics teaching in primary and lower secondary schools and early age programming education. He works in expert group for innovation of national informatics curricula at the National Institute for Education.



Z. Pluhár is assistant lecturer of Faculty of Informatics at Eötvös Loránd University, Budapest, Hungary. She is member of the T@T (Technology Enhanced Learning) Lab and works mostly in teacher education. She is the head of Professional Community of Public Education at John von Neumann Computer Society. Her research fields are computational thinking, education of robotics and STEM. Since 2011 she has been organizing the Bebras informatics contest in Hungary.



I. Pesek is assistant professor of Computers in Education at the Faculty of Natural Sciences and Mathematics, University of Maribor. He is involved in teacher training of CS teachers. His main research interest are in computers science education and use of ICT in education.

REPORTS

Cyprus Olympiad in Informatics

Panayiotis ERACLEOUS¹, Pavlos PAVLIKAS¹,
Adamos TTOFARI², Andronikos CHARALAMPOUS²

¹*Ministry of Education and Culture*

²*University of Cyprus*

*e-mail: paneracl@gmail.com, ppavlikas@gmail.com,
adamos2468@gmail.com, andronikos4796@gmail.com*

Abstract. This report presents the organization of the Cyprus Olympiad in Informatics in terms of the format used for each age group and the methods and tools utilized for the preparation and selection of the delegations of Cyprus for international competitions.

Keywords: Cyprus, International Olympiad in Informatics, Balkan Olympiad in Informatics, computer science, computer programming, curriculum, gymnasium, lyceum.

1. Introduction – Educational System in Cyprus

Education in Cyprus starts at the age of six and is divided into elementary and secondary education. Secondary education is divided into two levels – Gymnasium (ages 12–15) and Lyceum (ages 15–18). The Ministry of Education and Culture (MOEC), introduced Computers Science in Lyceums in the early 1990s. Computer Science was mandatory only in the first grade of the Lyceum, for two class periods per week. Students could choose to take the course in the second and third grade. LOGO and QBASIC were the programming languages used in the initial course.

A significant change was made in the early 2000s. Computer Science was made mandatory for all three grades of the Gymnasium. It was also introduced as an elective

course for the second and third grade of the Lyceum. Flow charts, Visual Basic and Pascal were the tools used to improve the algorithmic way of thinking of students.

Since then a lot has changed. In recent years, the CS curriculum in the public schools of Cyprus has undergone significant modifications in order to incorporate programming in each grade. Still, no CS curriculum exists for our elementary schools.

Currently, the tools and programming languages that are used for teaching programming in public schools are:

- Scratch (1st grade of Gymnasium).
- Alice, Robomind (2nd grade of Gymnasium).
- Pascal (3rd grade of Gymnasium).
- Pascal (1st grade of Lyceum).
- GameMaker (2nd grade of Lyceum).
- C++ (2nd–3rd grade of Lyceum).

The infrastructure in our Gymnasiums and Lyceums is relatively efficient. Every school has three to four computer labs, each one equipped with up to twenty computers. Every student works on his/her own computer during classes. A Computer Science teacher is responsible for the labs' maintenance and annual inspections are made to secure the longevity of the computers.

2. Cyprus Olympiad in Informatics (COI)

Cyprus participated for the first time in both IOI (International Olympiad in Informatics) and BOI (Balkan Olympiad of Informatics), in 1993. Cyprus Olympiad in Informatics (COI) was established in 2006 and it is organized annually by the Cyprus Computer Society (CCS) and the Ministry of Education and Culture (MOEC), in the following format. There are five academies, one in each district, that are responsible for preparing students for international competitions. The academies cover geographically most of Cyprus. In each academy, students are taught evening programming lessons, for two hours per week. Responsible for each academy is a Computer Science teacher assigned by the Ministry of Education. The top students are chosen to represent our country as members of the Cypriot delegations.

Up to 2011, the selection of the teams was made after two rounds of competition. The first round was on paper and the second one, although it was conducted with the use of computers, it was without the use of an online judge system and the solutions were judged manually. The main programming language the students used was Pascal.

In 2011 there was a major effort to improve things. C++ was introduced in the academies, as the main programming language and the curriculum harmonized with the one that is used in the IOI. Contest Management System (CMS) was used for handling the contests rounds. Another major change during that year was the introduction of the Bebras competition for younger students (12 to 15 years old). The Bebras contest runs in two rounds. The first is open for all Gymnasium students and the second is for the top students of each grade, which are also invited for the COI camps and lectures. In

Table 1
Cyprus medals in international competitions

IOI	BOI	JBOI	EJOI
• Silver (2017)	• Silver (1993, 2016)	• Silver (2017)	• Silver (2017)
• Bronze (1993, 2015, 2016)	• Bronze (1995, 1997, 2001, 2006, 2013, 2014, 2015, 2016, 2017)	• Bronze (2015 – 2 medals, 2016, 2017, 2018)	• Bronze (2017, 2018)

the past, most students participated in the COI competition after the age of 15. With the Bebras competition, younger students were attracted. The youngest contestant who competed in 2018 was 12 years old.

The results of the Cypriot delegations have improved drastically in recent years. Cyprus participated, for the first time, in the Junior Balkan Olympiad in Informatics in 2015 and won two bronze medals. Also, in 2015, Cyprus won its second medal in IOI after a long span of 22 years. As you can see from the statistics (Table 1), the changes have been effective:

2.1. COI Format

- Preliminary Round: Just before Christmas, four preliminary problems are announced publicly on CMS and students are given two weeks to submit their solutions. Their scores do not count for the first round.
- First Round (4 problems – 3 hours): Students that score at least 50% of the points of the first round qualify to the second round.
- Second Round (4 problems – 4 hours): The top 16~20 students qualify to the International Selection Rounds.
- BOI Selection Round (4 problems – 5 hours): Selection of the BOI team. Only students who have qualified from the second round can participate.
- IOI Selection Round (4 problems – 5 hours): Selection of the IOI team. Only students who have qualified from the second round can participate.
- JBOI/EJOI Round (4 problems – 4 hours): Selection of the JBOI/EJOI team. Only the students who participated in the second round and are eligible from JBOI age standards can participate.

Responsible for organizing COI is the Cyprus Computer Society (CCS) in co-operation with the Ministry of Education and Culture (MOEC). CCS is a professional and independent non-profit organization, seeking to improve and promote high standards amongst informatics professionals, in recognition of the impact that informatics has on employment, business, society as well as on the quality of life of the citizen. The MOEC is responsible for promoting Computer Science for all students in a unified educational system. The Ministry acts as the supervisor for COI in terms of personnel selection, support and setting a common policy for all districts involved.

3. Preparation for COI and International Contests (IOI, BOI, JBOI/EJOI)

3.1. COI Syllabus

The COI curriculum is divided into 3 parts, depending on which round the students are competing:

- 1) First Round:
 - Basic Programming.
 - Strings.
 - Arrays (1d, 2d).
 - Searching/Sorting.
 - Stacks/Vectors.
- 2) Second Round and JBOI Selection:
 - Functions/Recursion.
 - STL (maps, sets, queues, pairs).
 - Graph Theory:
 - Graph Traversal (DFS/BFS).
 - Shortest Paths (Dijkstra, Floyd-Warshall).
 - Minimum Spanning Trees (Prim, Kruskal).
 - Complete Search.
 - Greedy Algorithms.
 - Introduction to Dynamic Programming.
- 3) Curriculum for IOI and BOI Selection Rounds:
 - Advanced Dynamic Programming.
 - Bitmasks.
 - Advanced Graph Theory:
 - Trees.
 - Directed Acyclic Graphs.
 - Successor Graphs
 - Range Queries:
 - Segment Trees.
 - Binary Indexed Trees.
 - Sparse Tables.
 - Computational Geometry.
 - String Searching Algorithms:
 - Knuth-Morris-Pratt Algorithm.
 - Rabin-Karp Algorithm.
 - Tries.
 - Hashing.
 - Suffix Arrays.
 - Number Theory.

For developing the COI curriculum, an extensive literature review on competitive programming books was used as well as an in-depth investigation of other countries' preparatory systems.

3.2. Training Camps

In 2012, training camps were introduced. The focus was to teach the contestants advanced topics and give them an opportunity to meet each other and to build a learning community. The first years, the camps were two days long during Easter Holidays, just before the international selection rounds. The lessons were taught by guest lecturers from Greece. During the span of two days, the camps covered a lot of material, but because of the limited time, the content was not understood completely, by all students. Additionally, it was the only on-site advanced training our experienced contestants had during the year.

In 2018, winter camps were introduced and the format of the camps changed. There are two advanced topics lectures per day and a 2-hour contest at the end of each day, giving the opportunity to get hands-on the new knowledge acquired. Currently, there are two difficulty levels: junior and senior. The junior level was introduced to recruit new contestants to the competition and the senior level is for international contest preparation. The topics are different each time giving the opportunity for all contestants to learn something new in each camp they are attending.

In the future, there are plans to organize summer camps in order to train the national delegations, just before the international competitions. Moreover, lectures are planned during the summer for Gymnasium students in order to prepare them for JBOI 2020, which will be hosted in Cyprus and to better prepare our future IOI contestants.

3.3. Tools and Resources Used

These are the tools that are used within the lectures in order to prepare students for competing. These tools are used during the training camps as well:

3.3.1. Contest Management System (CMS)

We currently use CMS 1.4 as our contest environment for all the rounds of the competition. The setup is one machine that runs all the services and handles the submissions (log service, contest web server, admin web server, etc.). For BOI 2016, which was held in Cyprus, we used the CMS 1.3 and the setup was across three machines. The first machine was running only the services and the other two were handling the submissions, with four workers on each machine. No technical problems or delays in submissions were detected.

3.3.2 *Michanicos Online Judge*

Michanicos is a localized online judge build upon CMS and CMSocial, which currently holds approximately 200 problems in the Greek language and it is publicly available to our students. The platform allows for the district instructors to upload programming tasks and lecture notes through the administrator panel, set up different task tags and categorize their material. It allows for multiple contests to be run simultaneously, completely separated from the CMS platform used for our official competition rounds. The platform allows submissions in C, C++, Pascal, Java and Python, offers complete control over the students' submissions and generates reports and statistics for all contestants. Michanicos serves also as a repository for all the tasks used in our previous contests. It has been a very significant upgrade to our training process.

3.3.3. *Additional Resources*

Since the language barrier is not an issue as most of our students speak English fluently, the use of the following resources is highly encouraged.

Tools used for problem-solving training:

- Online Judges:
 - SPOJ.
 - Codeforces.
 - USACO.
 - CodeChef.

3.4. *Statistics*

The Table 2 shows the student participation in COI contests for the past five years.

4. Conclusion

Cyprus is a small island, with a population of 800,000 people. It is understandable that our selection pool is very small each year. Many of our students participate in other Olympiads as well (e.g. IMO, IChO). Some additional problems that COI is facing, is

Table 2
Student participation in COI and Bebras

Contest	2015	2016	2017	2018	2019
Bebras Contest	457	638	825	721	689
First Round	102	112	123	142	158
Second Round	43	52	67	62	44
Third/Fourth Round	12	15	16	20	21
JBOI Selection	17	16	24	14	13

the lack of academic support and the lack of training for our instructors. The COI alumni are very important too, but, unfortunately, most of them are studying abroad. In 2016, Cyprus organized the Balkan Olympiad in Informatics with great success, which we hope to repeat for JBOI 2020. Despite the problems, Cyprus puts a lot of effort and resources to help its students and put them in a position to be successful. We have gone from zero total points to a silver medal within the past eight years and we hope to win Cyprus' first gold medal in the near future.

References

- Bebras Cyprus. <https://bebras.org.cy/en>
- Contest Management System. <https://cms-dev.github.io/>
- CMSocial. <https://github.com/algorithm-ninja/cmsocial/>
- Cyprus Computer Society. <https://www.ccs.org.cy/>
- Cyprus Olympiad of Informatics. <http://www.coinformatics.org/>
- Dagienė, V., Skupienė, J. (2010). Olympiads in informatics: competitive learning of programming for secondary school students. *A New Learning Paradigm: Competition Supported by Technology*.
- Halim, S. and Halim, F. (2013). *Competitive Programming 3: The New Lower Bound of Programming Contests*. Singapore.
- International Olympiad in Informatics – Statistics. <http://stats.ioinformatics.org/>
- International Olympiad in Informatics – Syllabus. <https://ioinformatics.org/files/ioi-syllabus-2018.pdf>
- Kiryukhin, V., Tsvetkova, M. (2011). Preparing for the IOI through developmental teaching. *Olympiads in Informatics*, 5, 44–57.
- Kiryukhin, V. (2011). Method of carrying out and preparation for participation in the Olympiad in Informatics. *All-Russian Olympiad*.
- Kolstad, R., Piele, D. (2007). USA Computing Olympiad (USACO). *Olympiads in Informatics*, 1, 105–111.
- Michanicos Online Judge. <http://81.4.170.42:8980/training>
- Wang, H., Yin, B., Liu, R., Tang, W., Hu, W. (2010). Selection mechanism and task creation of Chinese National Olympiad in Informatics. *Olympiads in Informatics*, 4,



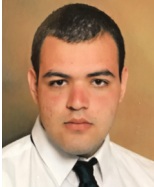
P. Eracleous works for the Ministry of Education and Culture in secondary education, since 2003. He holds an MSc in Computer Science from Middlesex University and a BSc (Hons) in Computer Science from Purdue University. He has been involved in curriculum development for secondary education and has co-authored the books used in Lyceum Computer Science courses. Since 2011, he is the Cyprus Olympiad in Informatics instructor for the district of Nicosia and he is responsible for the selection and preparation of the delegations of Cyprus for international competitions. He has been the team leader of Cyprus in IOI from 2015 to 2019. He was a member of the Scientific Committee for the Balkan Olympiad of Informatics hosted in Cyprus in 2016.



P. Pavlikas graduated from the Department of Electrical and Computer Engineering of the Aristotle University of Thessaloniki. He also holds an MSc in Information Systems from the Open University of Cyprus. He works for the Ministry of Education and Culture in secondary education since 2003. He worked as a Computer Science Advisor for the Ministry of Education for 5 years and he co-authored the curriculum and books used for secondary education. From 2009 until 2018 he was the COI instructor for the district of Larnaca. He has been the team leader for Cyprus in several international competitions.



A. Ttofari is a student in the Department of Computer Science at the University of Cyprus. He participated in IOI 2014 and IOI 2015 and he was a deputy leader of many Cypriot Delegations since 2016. He started as a teaching assistant in the district of Larnaca in 2015 and currently he is teaching in the Nicosia district, since 2016. Additionally, he serves as a problem setter and organizer for the training camps for COI.



A. Charalambous is a student in the Department of Computer Science at the University of Cyprus. He started as a teaching assistant for the district of Nicosia in 2016. He is in the technical committee of COI and he is responsible for the maintenance of the server machines and the administration of the Contest Management System.

The use of E-olymp Internet Portal in Programming Competitions

Mykhailo MEDVEDIEV

*School of Information Technologies and Engineering,
ADA University, Baku, Azerbaijan
e-mail: mmedvediev@ada.edu.az*

Abstract. E-olymp Internet portal was created to engage the students of higher education institutions and secondary schools to participate in programming contests, as well as to improve the quality of training future specialists in the information technology and programming areas.

Keywords: distance learning, programming competitions, online judge system.

1. Introduction

In times of transformations to information society, one of the nation's challenges is the training of specialists with relevant competencies, in particular preparing professionals in the area of computer science. Programming as a subject of study includes learning of various programming languages, algorithmic problems, program testing techniques, as well as work with data structures.

While training a specialist, distance learning systems play an essential role in the learning process, that includes sets of organizational, telecommunication, academic and scientific resources. Recently the distance learning has become a prominent part of the higher education system. It enables to select a convenient time, place and learning style for every student; improve their skills, acquire a profession in off-work hours; pursue higher distance education for people deprived of the possibility to receive traditional one. One of such systems dedicated to the distance learning programming is E-olymp Internet portal (www.e-olymp.com).

The Internet portal allows to facilitate the work of teachers and tutors in preparing for programming competitions, provide an opportunity for gifted students to work independently, to develop and exchange experience with like-minded students from different regions. The use of the portal provides an occasion to form and select qualified personnel who are able to arrange professional trainings for students – future teachers in the sphere of information technology, mathematicians and programmers.

E-olymp is considered to be a practice and a contest platform, like Topcoder, USA-CO, Codeforces, Codechef.

2. Functionality of E-Olymp Portal

E-olymp Internet portal was developed by Ivan Franko Zhytomyr State University, the Department of Applied Mathematics and Computer Science with the financial support of the Ministry of Education and Science of Ukraine in 2007. Over time, the functionality of the portal has been constantly improved. The portal administrators are:

- Mykhailo Medvediev – Assistant Professor at ADA University.
- Sergey Zhukovskiy – Senior Lecturer at Ivan Franko Zhytomyr State University.

The E-olymp portal helps the teachers of informatics and instructors of programming to deliver the elective courses and organize trainings and competitions. It allows pupils and students to prepare for olympiads independently, for instance, to solve thematic problems, to check their solutions without teacher participation, to compare the level of their knowledge and skills with the level of other pupils and students, which, in turn, creates ambition for victory and stimulates upgrading the skills and knowledge in this area.

So far, the portal supports four languages (Ukrainian, Russian, English and Azerbaijani), that allows to attract participants from different countries to programming competitions.

While developing the E-olymp system, the project team strived to make it user-friendly, fast and accessible to wide range of users. That is why the online judge system was made in the form of a website available on the Internet.

Nowadays E-olymp system supports the compilation of solutions in one of the following programming languages: Pascal, C/C++, Java, Python. It is possible to run personal or team competitions by ACM ICPC (International Collegiate Programming Contest) rules, as well as by IOI (International Olympiad in Informatics) rules. The overall rating of Internet portal users and participants of competitions is maintained. A forum with the opportunity to discuss competitions and individual tasks is supported. It is possible to form groups, where you can hold your own set of competitions. Precisely in the groups the distance summer and winter schools are held. In the groups the courses on programming languages, algorithms and data structures for university students are delivered.

The database information is processed using the Internet portal services. Any interested person can take part in competitions once registered in the system, or simply can check his solutions to the problems, which statements are available in the website database.

After testing the problem's solution, the participant's rating is recalculated. It is calculated using two parameters: the number of completely solved problems and the number of points scored. This is due to a different level of users and rules of official competitions. It should be reminded that by the rules of school programming contests,

the rating is calculated by the number of points obtained depending on the number of passed test cases. And by the rules of the student ACM ICPC competitions, the winner is the one who solved the largest number of problems completely. A problem is considered to be solved completely, if it has passed all the test cases proposed by the jury members. With the same number of solved problems, the time is taken into account. Penalty time is charged for every unsuccessful attempt.

During the portal activity since September 2009:

- More than 70,000 users registered.
- More than 9,000 problems uploaded to the portal.
- More than 5,000,000 submissions checked.
- More than 1000 user groups created.
- A large number of trainings and official competitions have been organized. For example, official competitions at schools and universities, regional and national competitions in Ukraine and Azerbaijan, mirrors of ACM ICPC and IOI programming contests, international student programming competitions, summer and winter programming schools (Sevastopol 2011–2013, Kharkiv 2009–2013, Qafqaz university 2011–2015).

In many universities of Azerbaijan (including ADA University) the following courses are offered to students using the groups of E-olymp system: Programming Principles (C/++, Java, Python), Data Structures, Design & Analysis of Algorithms, Object Oriented Programming. Groups enable the teacher to automate the verification process of writing programs by students. For example, every semester the author of this article teaches classes in 3 groups, each of them has more than 30 students. Each week, each student must solve about 10 problems. Thus, during a semester (14 weeks), the teacher should check for correctness about $3 * 30 * 10 * 14 = 12,600$ programs. This is not possible without automation!

3. Types of Problems Presented on the E-Olymp Portal

E-olymp portal contains large number of elementary problems in various topics, enabling to use them for a wide range of school teachers and university instructors in their basic programming courses. Exactly these problems teach the students to work independently: how to write properly the first program, how to enter the input data, what data type to use in the program, how to get the correct result. E-olymp system motivates the student in such work, because user immediately sees the answer of the online judge system after submitting the code (Accepted, Wrong Answer, Time Limit). Step by step, the student learns topics such as data input/output, linear programs, processing the digits of the number, conditional statement, loops, one-dimensional and two-dimensional arrays, strings, functions, bit operations, mathematical functions. Each of these topics in turn is divided into subtopics. For example, the set of problems related to the topic “linear integer array” with division into subtopics (problem numbers are given from E-olymp online judge) is given below:

- Elementary problems # 904, 7843, 7844, 8679, 8680.
- Sum of array elements # 919, 7829.
- Minimum / maximum in array # 914, 917, 928, 1952, 7831, 7832, 7845, 7849.
- Second minimum / maximum in array # 5059, 7834.
- Read array till the end of file # 8358, 8684, 8685.
- Average of array elements # 2238, 7368, 7833.
- Shift the array elements # 922, 4760.
- Reverse the array elements # 1460, 2098, 3935.

The next class of problems can be found at regional olympiads. In order to solve them successfully, students must have sufficient knowledge of basic algorithms from such topics as number theory, dynamic programming, combinatorics, computational geometry, graph theory, recursion, sequence processing, greedy algorithms, sort and search, string processing, data structures (Clifford, 2008; Cormen *et al.*, 2009; Weiss, 2012).

At national championships and international competitions (such as ACM ICPC or IOI), the knowledge of advanced algorithms is required. For example, one of the favorite topics at these contests is complex data structures and techniques for their processing, such as Range Minimum Query, Lowest Common Ancestor, Segment tree, Fenwick tree, Decart tree, SQRT decomposition. Below, for example, given the set of problems from the topic “segment tree” with division into subtopics:

1. Single update:
 - Sum of elements in given range # 2941, 4255, 4484, 4496.
 - Minimum/maximum in given range # 695, 2911, 3838, 4482.
 - Prefix/suffix sum # 2906, 2907, 4504, 4510.
2. Multiple update:
 - Sum of elements in given range # 1994, 2304, 2307, 2939.
 - XOR operation # 2905.
 - Additional data structures in the vertices # 3866, 5084.
 - Multiple operations # 752, 3984, 7761.
3. Persistent segment tree # 2955.
4. Two dimension segment tree # 861.
5. Dynamic segment tree # 3252, 7488.

Below we’ll review some problems of advanced level, for which solutions require to implement smart algorithmic ideas. The numbers of the problems are given from E-olymp system.

Problem #4516. Trees in the garden (www.e-olymp.com/en/problems/4516, *XX All-Ukrainian informatics olympiad*). The undirected weighted graph is given. The vertices of the graph are trees in the garden, the edges are paths between them. The garden was so large that one gardener was unable to take care of it. It was decided to divide the garden into two parts. Certain trees will be assigned to the first part, and the rest to the second. One part of the garden may remain empty. Each of two gardeners will take care of his part of the garden by walking along the paths connecting the trees of his part only.

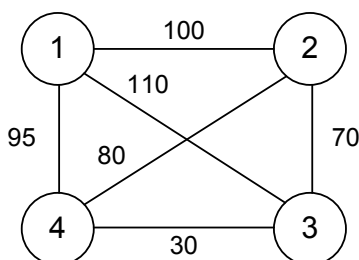
You must divide the garden in such way that the length of the longest path between a pair of trees belonging to the same part is minimum.

Solution. In this problem we need to assign to each tree one of the two gardeners who will take care of it.

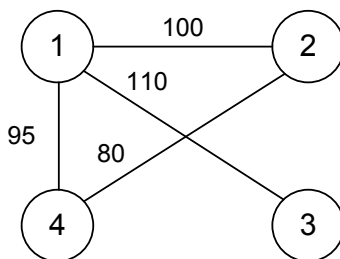
Let us consider the following subtask: is it possible to divide the trees between two gardeners in a such way that there are no paths (along which each gardener will walk separately) with the length greater than x . If we know the answer to this question, then we can search the minimum value of x (the answer to the problem), for which such partition exists using binary search.

Let us draw a graph where the edges of the length greater than x are only available. If such graph is bipartite, then each set of trees will be maintained by one of the gardeners. And none of the gardeners will have at their disposal the paths longer than x .

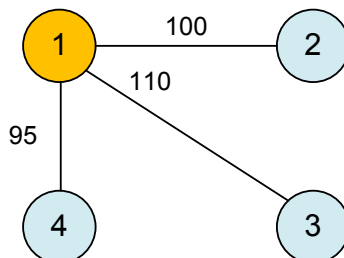
Example. Let us consider the following graph:



Let us try to divide the garden trees between two gardeners in a such way that there are no paths of length strictly greater than $x = 70$. For this purpose, take in the graph only the edges of size greater than 70 and check if it is bipartite:



This graph is not bipartite. Let us consider the graph with $x = 80$. This graph is bipartite:



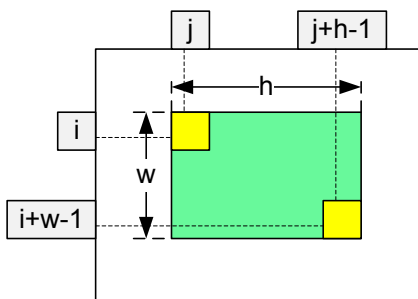
It can be observed that when $x < 80$, the graph with edges of weights greater than x always contains a cycle of odd length (1 – 2 – 4). Therefore, the answer is 80.

Problem #3535. Vasya and matrix (www.e-olymp.com/en/problems/3535). Vasya has a rectangular matrix with n rows and m columns with non-negative integers in the cells. For a given integer k find a submatrix of maximum area in which the sum of all numbers does not exceed k . The submatrix is a rectangular area of the matrix.

Solution. Let a be an input rectangular matrix. Let us create a two-dimensional array dp of the same size, where $dp[i][j]$ equals to the sum of numbers in submatrix with opposite vertices $(1, 1)$ and (i, j) . Its cells can be recalculated by the formula:

$$dp[i][j] = dp[i-1][j] + dp[i][j-1] - dp[i-1][j-1] + a[i][j]$$

Let us consider the submatrix of the array a of size $w \times h$. Let its opposite vertices have coordinates (i, j) and $(i+w-1, j+h-1)$.



Then the sum of all numbers in the submatrix equals to

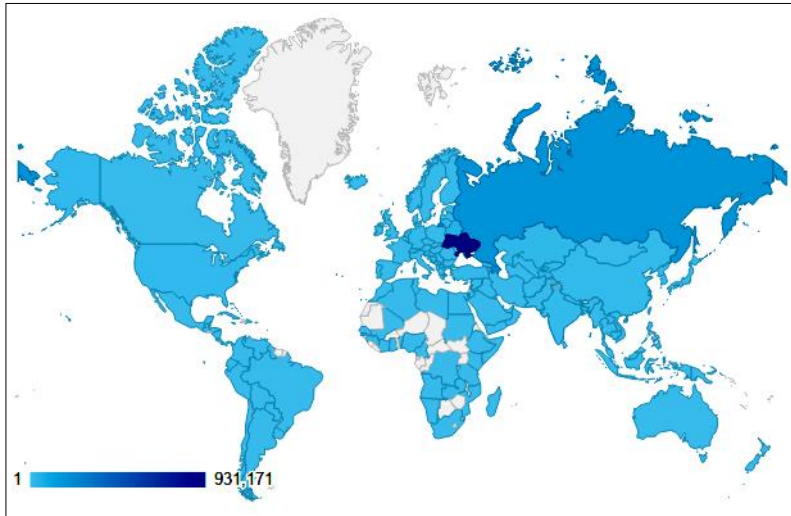
$$dp[i+w-1][j+h-1] - dp[i+w-1][j-1] - dp[i-1][j+h-1] + dp[i-1][j-1]$$

The problem is solved by exhaustive search. Let us run through all possible sizes of the submatrix $w * h$ and all possible left upper corners (i, j) of this submatrix. Within the constant time we can find the sum of numbers in this submatrix using the array dp . If the sum does not exceed k , then among all such submatrices we calculate the maximum of their areas. However, such search runs in $O(n^4)$ and can be improved to $O(n^3 \log n)$ using binary search by the width h of submatrix.

Let us fix the upper left corner (i, j) . Let $g(w, h)$ be the sum of numbers in the submatrix $(i, j) - (i+w-1, j+h-1)$. If w is fixed and h is treated as a variable, then $g(w, h)$ is a non-decreasing function relative to h (the array a contains non-negative integers). Using binary search we find the maximum value for h , for which $g(w, h) \leq k$.

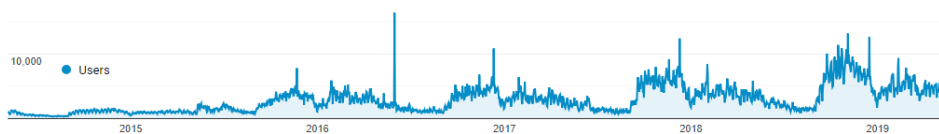
4. The Use of E-Olymp Portal

With the help of Google Analytics, we can see the statistics of problems submitted in the last three years:



The greatest number of students who use the E-olymp system are from Ukraine, Russia and Azerbaijan.

Over the last three years, the E-olymp system checks about 5,000 solutions every day. The peaks on the graph below correspond to online schools (competitions) when the large number of solutions are submitted. Minimums correspond to summer months, when students and pupils have holidays.



The following activities were run in groups of E-olymp system:

- Algorithms and data structures classes in Kiev branch of the Yandex School of Data Analysis, 2011–2015.
- Online summer and winter schools in Qafqaz University, Baku, Azerbaijan, 2011–2015.
- Summer school in Ukraine (Sevastopol) 2011–2013.
- Distance school in Ukraine “9 spiral turns” 2011–2015.
- Classes in programming languages, algorithms and data structures in Taras Shevchenko Kiev National University, 2009–2015.
- Lectures in programming principles 1 (C++), programming principles 2 (Java), data structures and algorithms in ADA University, Baku, Azerbaijan, since 2015.

The author of the article maintains the Facebook group “Competitive Programming Problems & Solutions”, where the theoretical material about algorithms is provided and training courses are run using the E-olymp system. Today the group has more than 2400 members.

5. Conclusion

The E-olymp portal is a convenient educational tool for preparing pupils and students for programming competitions. It can also be used in the further professional activities of a computer science teacher, encouraging to self-education and self-perfection.

References

- Clifford A. Shaffer (2008). *A Practical Introduction to Data Structures and Algorithm Analysis*.
Codeforces internet portal: <http://codeforces.com/>
Codechef internet portal: <https://www.codechef.com/>
Cormen, T.H., Leiserson, C.E., Rivest R.L., Stein C. (2009). *Introduction to Algorithms*.
E-Olymp internet portal: <https://www.e-olymp.com>
IOI – International Olympiad in Informatics: <https://ioinformatics.org/>
ACM ICPC – The International Collegiate Programming Contest: <https://icpc.baylor.edu/>
Topcoder internet portal: <https://www.topcoder.com/>
USA Computing Olympiad: <http://www.usaco.org/>
Weiss M.A. (2012). *Data Structures and Algorithm Analysis in Java*.



M. Medvediev is an Assistant Professor at School of Information Technologies and Engineering, ADA University, Baku, Azerbaijan. He received his Ph.D. degree in Computer Science from Kiev National University, Ukraine in 1999. His research interests include Algorithms and Data Structures, Competitive Programming and Education. He is a co-administrator of E-olymp online – judge system. He is a coach of ADA University teams at ACM ICPC since 2015.

TPS (Task Preparation System): A Tool for Developing Tasks in Programming Contests

Kian MIRJALALI, Amir Keivan MOHTASHAMI,
Mohammad ROGHANI, Hamid ZARRABI-ZADEH

*Department of Computer Engineering, Sharif University of Technology, Tehran, Iran
e-mail: {mirjalali, mohtashami, roghani}@ce.sharif.edu, zarrabi@sharif.edu*

Abstract. The task preparation system (TPS) is a tool developed mainly for preparing IOI tasks. It was originally developed for, and successfully used in IOI 2017, and since then, it has been used in several other nationwide and international programming contests, including IOI 2019. The tool consists of a command-line interface for local (offline) work, and a web interface which integrates with git and provides more features. This article presents the main features of the task preparation system, and briefly describes how it works.

Keywords: competitive programming, task preparation, Olympiad in Informatics, programming contest.

1. Introduction

The host technical and scientific committees of IOI 2017 had comprehensive discussions on how the development of IOI tasks could be eased. The first thing to fix was the directory structure of a task. Based on this convention, a set of scripts began to be developed to simplify the work. This finally resulted in a more mature software which is called TPS (task preparation system). Here are the feedback of two senior members of the IOI International Scientific Committee when presented with TPS, its documentation, and the tasks developed:

“That’s a great job you made. The system looks really nice and comfortable. My congratulations.”

“I must admit I’m also impressed that you’ve been able to develop a tool like this so quickly. This will surely be very helpful in organizing IOIs.”

Usage of TPS was not limited to IOI 2017. It has been used in many programming contests in Iran, including the national IOI team selection contests. The widespread usage of this tool motivated the authors of this article to share it with other members of the IOI community who are interested in designing and developing problems for programming contests.

TPS has a command-line interface (shortly, `tps-cli`) that is used locally by the developers to prepare tasks on their own machines. Each task is developed in a directory maintained through a git repository. A secured GitLab server was used in IOI 2017. The secondary part of TPS is the web component (shortly, `tps-web`) which is deployed besides the git repository in the server and provides additional features such as solutions invocation for more exact timings.

This article is organized as follows. We first describe the directory structure of the tasks in Section 2. We then explain the structure and features of `tps-cli` in Section 3. Finally, we briefly cover the main features of `tps-web` in Section 4.

2. Task Directory Structure

The TPS command-line interface acts based on a standardized directory structure which is usually maintained in a git repository for sharing with other task developers. The following files and directories are present in a task directory:

- **problem.json**: This is the main json file containing the general information about the task, such as short name, title, task type, memory limit, time limit, and possibly the `tps-web` URL if `tps-web` is set up.
- **solution/**: This directory contains the solution source codes, whether the solutions are correct or not.
- **solutions.json**: A json file containing an entry for each solution in the “`solution/`” directory, specifying the expected verdict of the solution on the subtasks, or whether it is a model solution (the one used for generating test outputs).
- **subtasks.json**: A json file specifying all subtasks by their names, scores, and validators assigned for verifying the conformance of their corresponding test inputs with their constraints.
- **validator/**: A directory containing codes for validating the format of test input files and checking their conformance with the constraints of the problem.
- **gen/**: This directory contains everything related to generating test data. The text file “`data`” in this directory specifies which tests are going to be generated by which generator and with what parameters. Manually created tests are also placed in the “`manual`” subdirectory. An example of “`gen/data`” file is shown in Fig. 1.
- **grader/**: For each programming language allowed in the contest, such as C++ and Java, there is a subdirectory here containing the graders for that language. A grader is a program that links with the contestant’s solution and provides it with grading interface say for reading the input and writing to the output.

```

@subtask samples
manual 01.in

@subtask n_3
general_graph    100    280    17    6
tree             98     5
@subtask n_2
@include n_3
general_graph    91555  40000  134   6
tree            10000   90
@subtask full
general_graph    100000  800000  543  44
tree            100000  800

```

Fig. 1. A sample “gen/data” file.

- **checker/**: A directory containing the checker, a program that evaluates the output of the contestant’s solution per test case and specifies its score.
- **public/**: This directory contains the files provided to the contestants, such as sample tests, compiling scripts, and basic graders for local testing.
- **statement/**: This directory contains the files related to the task statement.
- **scripts/**: The primary implementation of tps-cli commands is placed in this directory.

The following directories are also part of the TPS directory structure, but are not stored in the git repository:

- **tests/**: A directory containing the generated tests.
- **sandbox/**: Solutions will be compiled and executed in this directory.
- **logs/**: This directory contains the logs of the last execution of test generation or solution invocation.

3. TPS Command-Line Interface

The TPS command-line interface provides an easy way to execute predefined scripts commonly used during the process of task development. The scripts reside in the “scripts/” directory, and can be customized per need of a task. The “tps” command itself is placed in the PATH, and can be called anywhere in the terminal. For example, when the command “tps compile <arguments>” is issued, tps-cli runs the script “scripts/compile.sh <arguments>”. The following commands are currently supported in tps-cli:

- **compile**: Compiles a given solution with the appropriate grader, and leaves the result in the `sandbox` directory. It wraps the complexities of compiling solutions with different languages. The public version of the grader (the one given to the contestants) is used if `-p` or `--public` is passed to the command.
- **run**: Runs the compiled solution in the `sandbox` with no restrictions (like time limits). Input/output files can be specified through redirection.

- **gen**: Generates the test cases based on file “gen/data” and runs the corresponding validators on the inputs. The generated test cases are stored in the “tests/” directory. Multiple options are available such as specifying a subset of tests to generate (using wildcards), stopping on the first failure, and using an alternative model solution for generating outputs. An example of executing the command “tps gen” is shown in Fig. 2.
- **invoke**: Compiles and executes a given solution on the generated test cases considering the task restrictions (e.g. time limits). The score of the solution on each test case is also determined using the checker. Multiple options are available such as specifying a subset of tests to invoke, stopping on the first failure, and setting a different time limit. A sample execution of “tps invoke” is shown in Fig. 3.
- **make-public**: Updates the contents of the public directory (especially if the public graders are going to be generated through erasing the secret parts of the judge graders) and creates the archive provided to the contestants, based on file “public/files”.
- **verify**: Verifies the validity and integrity of the task directory structure including the json files.
- **analyze**: Opens the analysis page of the task in tps-web.

```

> tps gen
generator          compile[OK]
solution           compile[OK]
validator          compile[OK]
0-01               gen[OK]      val[OK]      sol[OK]
1-01               gen[OK]      val[FAIL]    sol[OK]
1-02               gen[OK]      val[OK]      sol[OK]
2-01               gen[OK]      val[OK]      sol[OK]
2-02               gen[OK]      val[OK]      sol[OK]
3-01               gen[OK]      val[OK]      sol[OK]
3-02               gen[OK]      val[OK]      sol[OK]

Finished.
>

```

Fig. 2. An example of executing “tps gen”.

```

> tps invoke solution/roads-test.java
solution           compile[OK]
checker           compile[OK]
0-01               sol[OK]      0.063       check[OK]    1 [Correct]
1-01               sol[OK]      0.069       check[OK]    1 [Correct]
1-02               sol[FAIL]    0.065       check[SKIP]  0 [Runtime Error]
2-01               sol[OK]      0.172       check[OK]    1 [Correct]
2-02               sol[OK]      0.076       check[OK]    0 [Wrong Answer]
3-01               sol[OK]      0.079       check[OK]    0.666 [Partially Correct]
3-02               sol[FAIL]    3.012       check[SKIP]  0 [Time Limit Exceeded]

Finished.
>

```

Fig. 3. A sample execution of “tps invoke”.

More information on the available options for the `tps` commands can be obtained by passing argument `-h` or `--help`. The source code and full documentation of `tps-cli` is available at: <https://github.com/ioi-2017/tps>.

4. TPS Web Interface

The TPS command-line interface is equipped with a web interface called `tps-web`. The web interface retrieves data directly from a task directory, parses the json files, and visualizes the task state to the users. As a result, users have a vision on all parts of the task and can efficiently apply different actions on it. Some of the main features of `tps-web` is briefly described below.

Task visualization. The web interface visualizes various components of a task. For example, one can see all the solutions for a task, and the verdict each of which should receive on the subtasks. (See Fig. 4.) As another example, there is a markdown viewer through which one can easily see the current version of the problem statement. (See Fig. 5.)

Solution invocations. The `tps-web` allows users to select a set of solutions, and invoke them on a subset of test cases. The invocation results are then presented in a table to-

Solutions				
#	Source Code	Language	Verdict	Download
1	akm-full-train.cpp	Auto-detect	Correct	+ Download
2	haghani-n3.cpp	Auto-detect	Runtime error n3: Correct n4: Correct	+ Download

Fig. 4. A sample solutions page in `tps-web`.

Problem Statement

Content

Mountains

Tahmuras, the third king of ancient Persia, has conquered a huge army of deevs (demons). He wants to imprison as many of them as possible in Alborz mountains and let the others go. Alborz is a mountain range with a skyline that looks like a polygonal chain with n vertices. The i -th vertex (for all $0 \leq i \leq n - 1$) has coordinates $(i, y[i])$, i.e. with longitude i and altitude $y[i]$.

Mountains

Tahmuras, the third king of ancient Persia, has conquered a huge army of deevs (demons). He wants to imprison as many of them as possible in Alborz mountains and let the others go. Alborz is a mountain range with a skyline that looks like a polygonal chain with n vertices. The i -th vertex (for all $0 \leq i \leq n - 1$) has coordinates $(i, y[i])$, i.e. with longitude i and altitude $y[i]$.

The deevs can be imprisoned on different vertices. No two

Fig. 5. A sample task statement page in `tps-web`.

Invocation details							
49 / 49							
	ac-saeed.cpp	adhoc-ceiling.cpp	greedy-saeed.cpp	optimized.java	roads-test.java	sub-k2.cpp	sub-k3.cpp
3-01	AC 0.083s/11MB	WA 0.026s/1MB	AC 0.091s/13MB	AC 0.362s/32MB	PS: 0.666 0.125s/11MB	WA 0.026s/1MB	WA 0.07s/9MB
3-02	AC 0.137s/7MB	WA 0.026s/1MB	WA 0.083s/7MB	AC 0.502s/32MB	TLE 1.733s/11MB	WA 0.026s/1MB	WA 0.051s/7MB
Max	0.137s/11MB	0.026s/1MB	0.091s/13MB	0.539s/32MB	1.733s/44MB	0.027s/1MB	0.07s/9MB

Invocation details on subtasks							
	ac-saeed.cpp	adhoc-ceiling.cpp	greedy-saeed.cpp	optimized.java	roads-test.java	sub-k2.cpp	sub-k3.cpp
k3	AC(4) Min score: 1.0	AC(2) WA(2) Min score: 0.0	AC(4) Min score: 1.0	AC(4) Min score: 1.0	AC(2) RE(1) WA(1) Min score: 0.0	AC(3) WA(1) Min score: 0.0	AC(4) Min score: 1.0
full	AC(2) Min score: 1.0	WA(2) Min score: 0.0	AC(1) WA(1) Min score: 0.0	AC(2) Min score: 1.0	PS: 0.666(1) TLE(1) Min score: 0.0	WA(2) Min score: 0.0	WA(2) Min score: 0.0

Fig. 6. An example of invocation in tps-web.

gether with their running time and memory used. An important feature here is the highlighting of the results which do not meet the expected verdict. An example of tps-web invocation is illustrated in Fig. 6.

Integrating with judging systems. Besides having an internal judging system for invocations, tps-web is also capable of integrating with other judging systems, such as CMS, in order to obtain an exact timing. It is in particular useful in programming contests such as IOI where the execution time of solutions are important up to milliseconds, and any small difference in hardware and software environment (such as sandboxing) can considerably affect the timing.

Discussion forum. In the time of task development, there are various topics that need to be discussed such as designing subtasks, reporting a bug, or an issue in the problem statement. In tps-web, every user can open a discussion about an issue and other users can reply to the thread.

Secure file sharing. During task development, there are numerous situations where developers need an easy way to share a file that is used temporarily during development. To address this need, tps-web provides a facility to store temporary files during development.

Export final packages. Since each judging system has its own directory format for presenting a task, tps-web provides the facility to automatically generate a package from the TPS directory structure. In particular, it has a built-in exporter for the CMS.

The source code and full documentation of tps-web is available at:
<https://github.com/ioi-2017/tps-web>.

Acknowledgments

We would like to acknowledge the complete list of people who were involved in developing TPS (in alphabetical order): Amirmohsen Ahanchi, Soroush Ebadian, Kiarash Golezardi, Ali Haghani, Keyvan Khademi, Hamed Saleh, Hamed Valizadeh, Mohammad Reza Maleki, Kian Mirjalali, Amir Keivan Mohtashami, Mohammad Roghani, and Hamid Zarrabi-Zadeh.



K. Mirjalali is a PhD candidate in the Computer Engineering Department at Sharif University of Technology. He was a member of the International Technical Committee (ITC) in IOI 2015 and also a member of the Host Technical and Scientific Committees (HTC, HSC) in IOI 2017. He won a silver medal in CEOI 2003 and was a world-finalist in ICPC 2007. He has been a scientific committee member of Iranian National Olympiad in Informatics (INOI) since 2003.



A.K. Mohtashami is a B.Sc student in the Computer Engineering Department at Sharif University of Technology. He was a silver medalist in IOI 2015. He was also a member of the IOI 2017 Host Technical Committee. He also has been awarded Asia West Championship in ICPC 2018 and was co-coach of Sharif's bronze medalist team in ICPC 2019.



M. Roghani is a B.Sc student in the Computer Engineering Department at Sharif University of Technology. He was a member of the Host Scientific Committee (HSC) in IOI 2017. He has received a Gold Medal in Iranian National Olympiad in Informatics (INOI) 2014 and a Silver Medal in Asia-Pacific Informatics Olympiad (APIO) 2015.



H. Zarrabi-Zadeh is a faculty member of the Computer Engineering Department at Sharif University of Technology. He was a member of the International Scientific Committee (ISC) from 2014 to 2015, and a member of the International Technical Committee (ITC) from 2015 to 2018. He was also the chair of IOI 2017 Host Technical Committee, the director of ICPC in the west Asia region Tehran site since 2012, and a bronze medalist as coach in ICPC 2019 world finals.

Computational Thinking in K-12: Azerbaijan’s Experience

Yahya TABESH¹, Shaya ZARKESH¹, Amir ZARKESH¹,
Ilaha FAZILOVA²

¹*Polyup, USA*

²*Innovation Technologies in Education, Azerbaijan*

email: yahya@polyup.com, shaya@polyup.com, amir@polyup.com, ilaha.f@ite.az

Abstract. Computational thinking is the process of finding numerical patterns and formulating algorithmic solutions. Polyup, a digital math playground, allows students to gain computational thinking skills through an experimental and gamified environment. Azerbaijani schools tested Polyup in their classrooms to see if it improved student attitudes towards math and motivated students to practice their math abilities. In this paper, Polyup is presented, the methods of deployment and usage of Polyup are reviewed, and we summarize the impact that Polyup has had on Azerbaijani students and schools.

Keywords: computational thinking, educational games, math education, Edtech, Azerbaijan.

1. Introduction

Computational thinking, an important theme of computer science for K-12 education, is defined as “the thought process involved in expressing solutions as computational steps or algorithms that can be carried by a computer” (NCSM, 2018). The importance of computational thinking extends beyond computer science and is a fruitful pathway to build the conceptual understanding of problem-solving and algorithmic thinking.

How can we develop an effective learning environment for computational thinking education in K-12? Mathematics educators have long seen the value in utilizing aspects of computer science to support the learning of mathematics, and computer science learning will in turn benefit through such a process. Therefore, computational thinking can be seen as a catalyst in learning mathematical concepts by utilizing the tools of computer science.

Which concepts of computer science can be used as such tools? Mainly algorithmic thinking through computing and approximation, using variables as identifiers and

containers, loops and iteration, boolean operators, conditional loops, and recursion. All these tools of computer science can be taught in the language of mathematics rather than a specific coding syntax. We present such a computational thinking “playground” where students can experiment with these building blocks and create expressions and algorithms.

2. Computational Thinking Playground

Computational thinking is a four-stage problem-solving framework consisting of decomposition, pattern recognition, abstraction, and algorithm design, as shown in Fig. 1. We have enriched and connected the stages with a “playground” as an environment for the experimental problem-solving.

We intend to present a digital computational thinking playground that has been used by more than a hundred thousand K-12 teachers and students as a platform for problem-solving. This playground is an easily accessible place where learners can tackle problems through experimentation.

To build this playground, we used a functional programming environment as the medium for problem-solving. A solution to the problems in this environment involves executing a sequence of functions. Functional programming treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data, so it is a powerful tool that can be used in a modular form for problem-solving. Such modularity is key, as it specifically empowers learners to utilize what they have built in the past for future solutions.

The proposed computational thinking playground empowers learners in reasoning, problem-solving, and algorithmic thinking in a gamified fashion. Anonymous user data is gathered from every learners’ moves, providing a very rich platform for learning design research. Results of the analyzed data can be used to improve the platform and also bring recommendations and feedback to the learners (Tabesh, 2018).

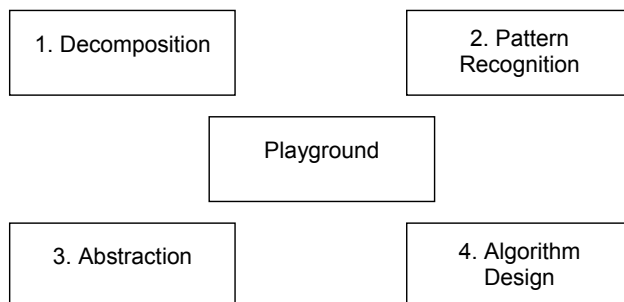


Fig. 1. Four Steps of Computational Thinking.

3. Polyup Platform

We considered the following objectives for the platform:

- Enable creative engagement.
- Develop mathematical skills.
- Support a growth mathematical mindset.
- Be collaborative and social.
- Equity and Accessibility.

The developed computational thinking platform is called Polyup (Polyup, 2019) and is a web application enabling problem solving in a functional programming paradigm. By using the elements of mathematics as the building blocks of the paradigm, students can be on-boarded quickly and spend more time learning how to solve computational problems rather than use the particular platform.

In the platform, the user is equipped with numbers, operations, and basic functions. The user can create a stack of computation in a functional modular form; computation simply goes top to bottom in a postfix style. Users can drag and drop numbers and operations on stacks to build an algorithm that achieves each puzzle's desired output.

The Polyup platform includes numbers, operators, booleans, variables and functions, as shown in Fig. 2.

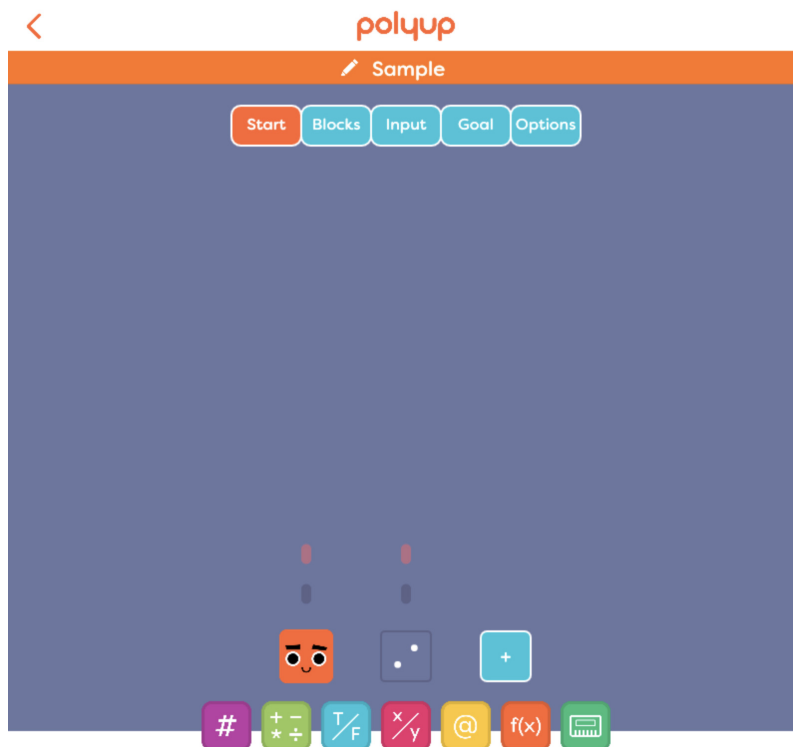


Fig. 2. Polyup Platform blocks.

The following features can be found on the Polyup platform:

- Reverse Polish Notation (RPN).
- Variables as identifiers.
- Iterative processes and loops.
- Boolean functions and conditional loops.
- Recursion.
- Block programming.

To learn more about the Polyup platform, we look at the following toy examples to calculate $3 + 4$ in iterative (Fig. 3.a and Fig. 3.b) and recursive (Fig. 4.a and Fig. 4.b) fashions. Of course, we could just write a 3, 4, and a plus sign, but we use these round-about approaches as a window into more advanced algorithms that can be created.

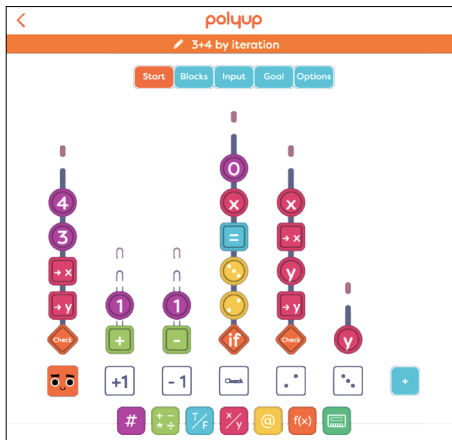


Fig. 3.a: Script of $3+4$ by iteration.

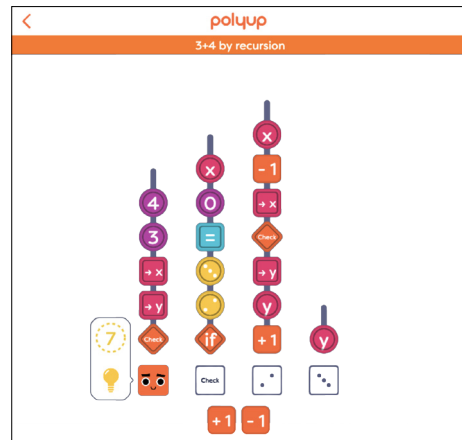


Fig. 3.b: User interface of $3+4$ by iteration.

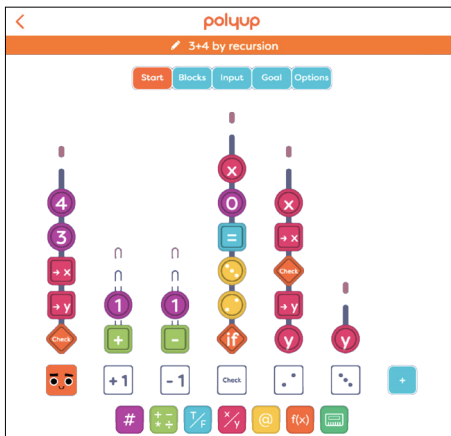


Fig. 4.a: Script of $3+4$ by recursion.

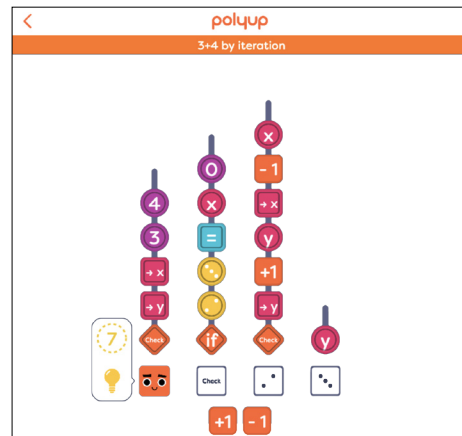


Fig. 4.b: User interface of $3+4$ by iteration.

In the iterative approach, we first set our variables, x and y , in the first stack. 3.a shows the puzzle in creation mode -- here, the puzzle's creator gets to define what the player sees at the start of the puzzle, the goal of the puzzle, and the blocks the player gets to use to achieve the goal. The author chose to give most of the structure of the program to start with, but notice that +1 and -1 stacks are modularized and hidden from the player in Fig. 3.b (the stacks have a dashed line around them in 3.a, meaning they are hidden stacks). Fig. 3.b shows the player's solution. When the player runs this program, Poly goes from top to bottom, and the variable "set" blocks take in the block directly above them. As a result, x will be initially set to 3 and y set to 4. Then, we go to the "Check" stack to check if x is equal to 0. If it is, then we return y , which represents the sum so far. Otherwise, we increment y by 1 and decrement x by 1. So, when computing, y will be incremented and x will be decremented x times, leading the final result to be the final value of y , or $4 + 1 + 1 + 1 = 7$.

Alternatively, in the recursive approach, we call the "check" stack over and over while decrementing x until it hits 0. As a result, the stack denoted by the two dots will be called x times, and thus after all the calls, we will have a stack of x "+1" blocks, which will in effect increment y by x , and return x plus y .

4. Polyup in Azerbaijani Schools

Polyup was introduced in Azerbaijani schools in October 2018. In order to encourage the use of Polyup's platform in Azerbaijani schools, with the aim of developing students' computational thinking and involving them in the game-based learning environment, the following steps have already been implemented:

1. Recruitment of Master trainers, localization, and development of training programs.
2. Content development aligned with the local math curriculum.
3. Presentation of Polyup at education fair, conferences and teacher workshops.
4. Pilot implementation in selected schools.
5. Involvement of schools, students and teachers to the Azerbaijan Challenges within Polyup.

Brief discussions about each of the above items are as follows:

1. Nine selected local master trainers were trained over video by the Polyup team. These master trainers in turn translated the teacher's guide and adapted training program materials to their local curricula. They provided online support as well as face-to-face training. However, on most occasions face-to-face meetings were held with teachers, as per their request.
2. Master trainers identified topics that were not mentioned in the portal and new machines were created and placed in the portal for 15 projects in 5 topics for grades 1 to 2, 11 projects in 11 topics for grades 3 to 5, 6 to 8, 9 to 12 and 9 projects on agricultural technology and green (alternative) energy. These machines, consisting of 3-7 chips, were highly demanded by teachers and students and were

actively used during the lessons. Additionally, the local Polyup team localized and adapted the “Teacher’s Manual” on how to use the platform and create new machines.

3. The Polyup Platform was presented at the Education Fair organized by the Ministry of Education and at the AgTech and Green Energy Forum, jointly organized by the Ministry of Agriculture and the Ministry of Energy. The benefits and advantages of Polyup platform were presented and discussed during 30 workshops and roundtables, both in Baku and the various regions.
4. In order to promote the platform, the local Polyup team organized school visits and conducted information sessions for school principals and teachers. The interested teachers from more than 14 schools were trained and registered on the platform. Throughout the training, teachers were informed about the principles of working with Poly Machines the creation of new Machines. Moreover, teachers developed their skills to create project-based machines. As a result of training program, teachers and students were engaged and actively participated in the Azerbaijani and international challenges established by Polyup. The methodology for using Polyup online platform during lessons is as shown in Table 1:

According to the observations, the Polyup online platform is mainly used during lessons in the following forms (see Table 2):

5. During the info sessions and pilot implementation, all teachers were also introduced to Azerbaijan Poly Challenge. They were trained on how to join the Challenge. Starting from March 2019, five Azerbaijan Poly Challenges were held and five winners were awarded with different prizes, such as tablets and notebooks. Moreover, one of the project participants, who joined the international challenge “Youcubed Prize”, has won a chance to participate in online course for math teachers organized by Stanford University. More than 200 teachers and 1000 students from Azerbaijan participated in the Azerbaijan and international challenges.

According to the results of a survey conducted by local Polyup team students, Polyup project raised interest and competitiveness among them, since the platform is focused on game-based learning. Therefore, with the support of the Ministry of Education, more schools, students, and teachers are expected to be involved in the future.

Table 1
The methodology for using Polyup

№	Methodology	Recommendations
1	Practical lessons	Can be used as mathematical calculations in the process of teaching STEM subjects
2	Project-based lessons	Can be used as integration with social sciences in the form of long and short-term projects
3	Problem based lessons	Can be used to connect mathematics with real-life experiences

Table 2
Polyup online platform using during lessons

Nº	Type of activity	Use form	Used tools	Recommendations for use
1	Class	Teacher assigns a class work. Students fulfill their assignments. Then, the student who quickly finished the assignment performs the solution of task on the machine on the smart board and sees if the sequence is correct	Smart board, computer, projector	The platform can be used during lessons (topic explanation, classroom assignments) for classroom exercises and after-school activities. All students are involved in the discussions and cooperate through the activities
2	Small groups	Teacher gives an assignment to small groups (consisting of 4-7 students) of students. The groups solve tasks on the computer	Computers, projector	Through this type of activity, students are involved in a competitive environment, which further engages the students
3	Pairs	Teacher gives an assignment to the students. Students solve a task on the computer in pairs	Computers, projector	Through this type of activity, students are involved in a competitive environment, which further engages the students
4	Individual	Teacher gives an assignment to the students. Students individually solve tasks on their computers	Computers, projector	The activity identifies the individual potential of each student, so a teacher can guide them accordingly

Reference

- NCSM (2018). *Computer Science and K-12 Mathematics*, NCSM.
- Polyup Casual Modding Platform* (2019). <http://www.polyup.com>. Accessed May 1, 2019.
- Tabesh, Y. (2018). *Digital Pedagogy in Mathematical Learning*, Invited Lectures from the 13th International Congress on Mathematical Education, ICME-13 Monographs, Springer.



Y. Tabesh Co-founder of Polyup. Visiting Professor at Stanford University. Distinguished faculty at Sharif University of Technology. Erdős award winner 2010.



S. Zarkesh Co-founder of Polyup. Student at the University of Pennsylvania Studying Math, Computer Science, and Business.



A. Zarkesh. Co-founder and CEO of Polyup. Serial entrepreneur with a background in teaching and education.



I. Fazilova Computer Science Teacher. Program manager - Innovation Technologies in Education.

Kids Programming Marathon: A Step toward Better Engagement with Computer Science Education

Maya TAKI¹, Ammar ALNAHHAS²

¹*Syrian Virtual University, Damascus, Syria*

²*Faculty of information technology engineering, Damascus University, Damascus, Syria*

e-mail: maya.taki@gmail.com, a.alnahhas@damasuniv.edu.sy, eng.a.alnahhas@gmail.com

Abstract. Due to the importance of spreading computer science education among young people, we present in this paper our work in preparing and organizing a computer science competition for children from 8 to 15 years old, named Kids programming marathon, the marathon goes in three phases and targets all kids in the country, tasks of the marathon are divided into three different types, each type is intended to support different skills for children, we show our motivation and goals of the marathon, we present the process of the marathon in details and show the materials of the competition, how it was chosen and how it is used in the tasks, we show some statistics, and finally discuss the impact of the marathon on the society, and our view for the future of this competition.

Keywords: computer science education, competition, young people, programming marathon.

1. Introduction

As the World Development Report for 2019 (*World Development Report 2019: The Changing Nature of Work*, 2019) stated that: "The nature of work is not only changing – it's changing rapidly. We don't know what jobs children in primary school today will compete for, because many of those jobs don't exist yet. The great challenge is to equip them with the skills they'll need no matter what future jobs look like – skills such as problem-solving and critical thinking, as well as interpersonal skills like empathy and collaboration, and the most effective way to acquire these skills is to start training at early ages."

We live in a world that is rapidly evolving, with technology tightly intertwined in life, in school and at work. Learning computer science (CS) helps people better understand our technology-enabled world. It positions students for high-demand jobs and provides them with skills that are broadly applicable – illuminating new approaches to problem-solving, critical thinking and creativity.

Technology can be a powerful force for social and economic inclusion and for addressing the many challenges facing our communities. By empowering children at early ages, we're investing in building stronger and more resilient communities.

Due to the importance of early learning of analysis and logical thinking skills for children (*2018 State of Computer Science Education*, 2018) and the role of competitions in introducing computer science in more interactive way, couples of years ago we worked with the Syrian Virtual University, the organizational team of the SCPC – the Syrian Collegiate Programming contest and other scientific partners to launch the kids programming marathon (KPM), an annual programming competition for children and adolescents in many regions and cities in Syria.

The Kids Programming Marathon is considered as an updated version of a similar competition that was part of the Syrian Olympiad in Informatics (SOI) (ALNAHHAS & ALAZAB, 2015; Idlbi, 2009) which was held ten years ago, and was halted in 2012, when the national Olympiad was restricted to secondary school students, due to the war conditions in Syria.

Compared to other competitions related to computer science and programming which are held either locally or internationally (Scratch Olympiad, IOI, Bebras) our training materials cover many aspects like logical problems, visual programming, and textual programming, in a more useful mix in introducing Computer Science concepts at early ages. Visual programming serves as a brief introduction to programming aspects like loops, conditions and variables in more interactive way and without Syntax restrictions. On the other hand, textual programming serves a more professional way in programming that is compatible to problems in the IOI. Logic questions are needed to introduce computer science concepts in a very familiar way like problems in Bebras competition. Integrating these three aspects brings more learning benefits to contestants than just focusing on one aspect.

This paper demonstrates our main objectives and motivation in section 2, in addition to the materials in section 3, statistics we worked on are presented in section 4, and the scientific and administrative improvements we made to our previous work in SOI, section 5 discuss the future of the event and its impact, and we conclude the paper in section 6.

2. Background and Motivation

In recent years, computing has grown in our daily lives. It is no longer confined to commercial or industrial applications, but extends to all aspects of our activities in life, in school and at work. As computing becomes more important, children at schools became more interested in learning how to create new technologies other than just using word processing or spreadsheets.

Computer science education or “CSE” is a very large subject. It blends all the “STEM” subjects of science, technology, engineering and math, and also includes design. It's important for students to learn these skills because computer science is everywhere.

By increasing access to CS for all youth as early as possible, we help them prepare for the jobs of today and tomorrow. This education gives them the opportunity to become the world's next innovators.

To accomplish this fluency, we need to deal more seriously with the syllabus of computer science, and its important aspects of learning writing in addition to reading, in the language of the computer: learning programming and the accompanying learning of the basic logical and mathematical concepts contained within its practices.

Competitions are one of the most important ways to encourage young people to discover new fields. They have been widely used to introduce children to various fields of science, including computer science that has algorithmic nature, in which the emphasis is on testing problem solving skills and logical analysis through an entertaining and rich experience in which the contestant learns programming the computer to serve its interest. Therefore, we are working on organizing a programming marathon for children and adolescents, to raise the general scientific level and enhance the skills of analysis and creative thinking among all.

2.1. Main Objectives

- Support teaching of computer science at schools, raise the general scientific level and enhance the skills of analysis and creative thinking for all children.
- Support the community and provide useful content in the field of programming and informatics and the development of fluency in dealing with information technology.
- Encouraging outstanding students in the IT field.
- Support the youth ability to come up with solutions to problems that are facing their societies, which is considered as economically – disadvantaged after a long war.
- Programming training will serve as a learning model, demonstrating how computer science education with informal learning settings can support the development of technological fluency.
- enabling youth people to design and create projects that are meaningful to them and their communities (Hubwieser, Armoni, & Giannakos, 2015).
- Training and preparation at early ages which is significantly reflected on the performance of teams participating in the Informatics Scientific Olympiad later.

2.2. Participation and Problems' Categories

The Kids Programming Marathon includes two competitions:

- The junior competition (8–11 years).
- The senior competition (12–15 years).

The two competitions aim to test the optimal performance of solving several questions divided into three categories:

- A problem formulated as a game, required to be solved using the graphical programming language Scratch, which offers an enjoyable start and easy to identify the concepts of programming.
- Computational and Logical thinking Problems.
- C ++ scripting problems, taking into account the complexity for each age group.

2.3. Structure and Stages

The Kids Programming Marathon plan consists of three phases:

- **Phase 1:** The first qualification test based on logical tests for all applicants, logical questions will be in the form of multiple-choice for all categories. All kids in the country are eligible to participate in this phase, kids should solve 20 to 30 questions with equal marks for each using a special system on computer. Participants should apply in special contest centers prepared for the competition, one in each province or area, so students are assigned a nearest geographical center to his residence area. Registration is held online some weeks before the competition. The goal of this phase is to filter the participants to find who are suitable for the contents and the objectives of the marathon, so all kids with 40% and higher of full mark are qualified to the next phase. An optional training program is offered to the qualified students with the help of Syrian virtual university centers and other sponsoring organizations and institutes.
- **Phase 2:** The second qualification test, based on tasks that cover logical and programming tests for all candidates from the first phase. Contest are also conducted using a special computer system and is held in special centers where each participant is assigned to the nearest center. The competition consists of Five multiple-choice logic tasks -which are corrected automatically-, one or two scratch tasks and two textual programming simple tasks, both Scratch and textual programming problems are corrected manually by the scientific committee (details in materials section). Qualification of this phase is based on choosing fixed number of students with higher total marks. The winners constitute the participants of the marathon finals.
- **Phase 3:** The final competition for all the candidates from the second phase, the contest is held in a single center in the capital where all participants from all other cities should gather. The contest is held using the special computer system where logical tasks are corrected automatically and programming tasks are corrected manually as will be described in the materials section of this paper, tasks are distributed amongst three types of the marathon, with about ten logical tasks, two scratch tasks for first age division and one for the second division. First division tasks consist of simple output-only tasks whereas second division students should solve two tasks using C++ programming language. The top three students of each division are the winners of the competition.

3. Materials

The goal of the marathon is to stimulate creativity, logical thinking and problem-solving capabilities of children and adolescents, so the material should be chosen carefully to achieve these goals. We decided that the tasks should be in different types, each type should be related to a target that leads to one of the general goals, besides the mixture of tasks should be consistent and should contribute to the general aim of the marathon.

Three main types of tasks are used with different mark distribution for each division of participants as will be discussed later.

- The first type is **Logical Thinking Tasks** that aims at capturing the creativity and innovation skills of the participants.
- The second type is **Scratch Tasks**, where Scratch is a well-known programming tool for young people that helps interactively program games, stories and animations (Resnick, Kafai, & Maeda, 2005).

The aim of these tasks is to teach kids problem solving skills without the need of teaching them advanced syntax of textual programming languages.

- Third type of tasks is **Textual Programming Tasks** that target advanced problem-solving skills including solution analysis, synthesis and testing.

3.1. Logical Thinking Tasks

These tasks are used mainly in the early phase of the marathon, whereas a different form is used in the proceeding phases, the advantage of this type of tasks is that it does not need any prior training, it is very useful to test logical thinking and deduction abilities of children. In the first stage a very simple form of this tasks is used, this form is similar to the one used in IQ tests, but it should be tolerated to be easily understood by young people, the target of using such tasks is to measure the logical and mental capabilities of kids, the following is a sample question of this type:

To prepare a pie we need 3 apples and two oranges, if we have 10 apples and 10 oranges, how many pies can we prepare?

A: 1
B: 3
C: 6
D: 10

Tasks of this form can be mathematical, logical or linguistic; which allows to measure different aspects of child skills.

In the second and third phases of the marathon a different form of logical tasks are used, which are Bebras-like tasks, in which a story with brief description is provided, and a multiple-choice question is to be answered, children should analyze the content and find the suitable answer, this kind of questions are more suitable for second division where children are older, but we find that they can be used for younger children as they

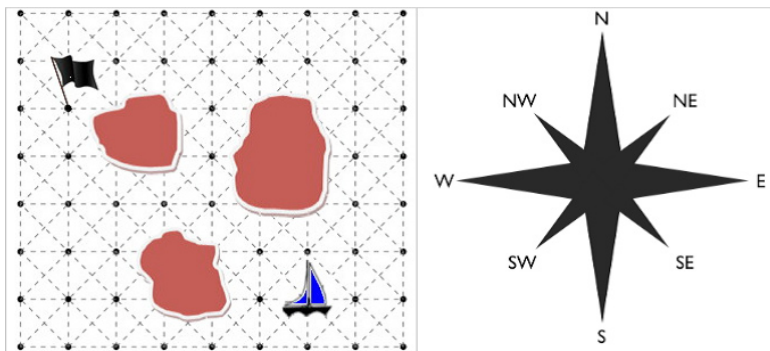


Fig. 1. Sample of used Bebras task in the marathon, the task is about finding a shortest path between the boat and the flag.

accepted and many of them managed to solve them in the last year. The aim of this form of tasks is to measure the analytic thinking of the kids along with the ability to comprehend and perceive the content to get the correct answer. Fig. 1 shows an image of a Bebras task that is used in the last year final stage of the marathon, a convenient story is provided to refer to the task of finding the best path.

In our first experiment which was a part of SOI in 2007, we used a different content for this type of tasks, they were about logical circuits, number theory and logic, but we find later that these subjects are no more suitable as they do not reflect skills but just knowledge, besides they are now more popular and are taught at school. Whereas the new model of IQ style and Bebras-like tasks are more convenient as it is shown by our new experiment.

Tasks of this type are in multiple-choice form and are always automatically corrected by special contest computer system designed for the marathon.

3.2. Scratch Tasks

Scratch is a well-known visual programming language for young people; it uses the principle of interactive programming to help users make games and animations. Tasks of this type is intended to be with algorithmic background in the marathon. So, we use this type of tasks to measure and teach problem solving skills to children. That means we are investing the programming part of Scratch rather than animation and movement parts.

To illustrate this idea, we show a sample task that has been used in the marathon before; Fig. 2 shows a sample image of Scratch task, in this task the child should program the movement of the characters as given in the task statement, he should process the interaction between the items and design the movement algorithms accordingly. We try to format the task in an approach where children should not rely on static knowledge of Scratch environment, instead the key point of the task needs to be designed according to the problem-solving abilities, that can be defined by using the appropriate item in the

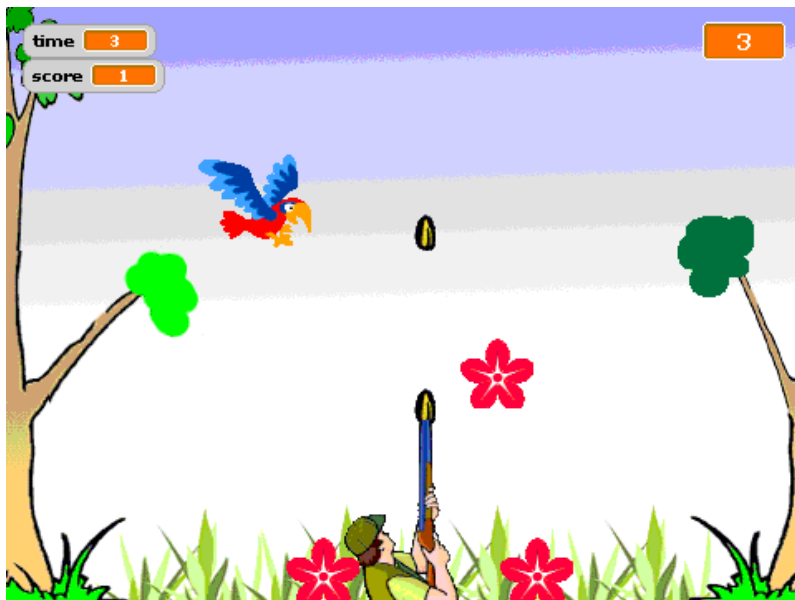


Fig. 2. Sample of Scratch task, a game where the child should develop a suitable logic for it.

correct context, rather than just memorizing and understanding the use of each tool, this way the creativity of the children is measured correctly, besides; we design the task to have a trick that needs mathematical or algorithmic solution. Experiments held from 2007 to 2011 showed that this type of tasks was very attractive for children and helped distinguishing creative ones despite the poor Scratch tools at that time. As Scratch now contains the concepts of lists and function it is now more suitable to design more creative tasks that are better to distinguish kids with high problem-solving abilities, results of the last two years competitions revealed that this type of tasks are more favorable among younger children, older children in the second division tends to prefer textual programming tasks as will be discussed later.

Scratch tasks are corrected manually by scientific committee members, the task is divided into subtasks, each subtask is assigned a portion of the total mark, where this portion is given if the subtask is totally achieved with no partial marks.

3.3. Textual Programming Languages Related Tasks

We believe that even if visual programming can measure creativity and problem-solving skills, there is still a necessity to include this type of tasks even for the younger children of the first division. There are many forms of tasks in this type, depending on the phase and the division,

For first division it is very difficult for kids to compose a working program, yet they can understand the syntax and comprehend the different items of programming

language, therefore the best task form for them is to find the output of a given program, this task allows us to measure kids knowledge in the language syntax but it is tough to capture problem-solving skills by this method for children younger than 12 years old. This type of tasks that rely on finding the output is corrected manually; each part of the output is given a portion of the total mark of the task, and is given if the output is correct.

By the other hand, textual programming tasks including ones where a complex algorithm is needed proved to be very attractive and suites adolescents in the second division, they preferred it over visual programming tasks and tend to find it more challenging.

Tasks are prepared in a way that suites ages between 12 and 15, very sophisticated algorithms are avoided and some tasks is designed to measure programming skills rather than algorithmic experience of the competitors. Tasks are corrected manually, because part of the marks is allocated for code analysis, the task is first evaluated in a way similar to IOI style where output should be correct with specific time and memory limits, but if the task fails to generate correct output the code is examined manually and part of the mark is given if code reflects correct algorithm.

The most distinctive part of the marathon is the mixture of the above three different types of tasks, the tasks complement each other and integrates together to fulfill the target of the marathon by encouraging all different skills of kids as well as discovering creative and distinguished ones, the mark ratio of each type is chosen according to the age division and with accordance to the goals, for the first division, Scratch tasks cover 65% of the competition mark, 20% for logical tasks and 15% for textual programming, leaving a wide range for visual programming and keeping the advantage of including logic and textual programming. For the second division 35% of competition mark is for Scratch, 20% for logic and 45% for textual programming, these ratios are chosen as we notice that adolescents in this age prefer textual programming to visual ones.

4. Statistics

The old competition held from 2007 to 2011 proved to be very impressive, about 70% of the Syrian medalists in IOI from 2012 to 2018 were winners of that competition.

As the mark distribution among task types is elaborated in the last section, along with the correction and mark assignment scheme, Table 1 shows the average results of the first division for the last year, there was 5 logical tasks with marks distributed equally, two scratch tasks with equal marks and four textual programming tasks in a type discussed in the material section.

Table 1
Average marks for first division

Logical tasks	Scratch	Textual Programming	Total average
50.65%	33.8%	36.84%	37.63%

Total number of participants in the last phase was 75, we can notice that the logical tasks get high average whereas other types are around the total average, which reflects the need for this type of tasks in this age division. Textual programming tasks are pointless as the type of tasks for this division is not creative as mention earlier in the material section.

Table 2 shows the average results of the second division, there were four logical tasks, one scratch task and two textual programming tasks, with marks distribution mentioned in the previous section.

It is clear that logic tasks have a very high average, this is due to the fact that all students in the last phase of the competition are high skilled, It is normal that programming average is low, it is still difficult for children to compose a full working program with textual programming languages, actually, the average is very promising as the evaluation of the tasks is similar to the IOI style with multiple test cases with small part of the mark allocated for code analysis as mentioned earlier.

An important point to notice is the gender distribution of the participant, Fig. 3 shows this distribution. There are many points to consider in this statistic, firstly the number of females is relatively high compared to average female participants in computer science competitions, this is due to the involvement level of family members for young aged children. The most important point to notice is that the ratio of female participants decreases in the second division, and taking into account that the number of female participants in the national Olympiad for secondary school students is very low, this indicates that the key factor of increasing females involvement in computer science competitions is to encourage them to participate as they are younger starting from the age of 8 and provide more motivation and encouragement for them to go on. To achieve this, we grant special awards to female participants in the marathon to motivate them to stay in the track and to attract more females to participate in the future.

Table 2
Average marks for second division

Logical tasks	Scratch	Textual Programming	Total average
80.17%	41.63%	21.55%	41.31%

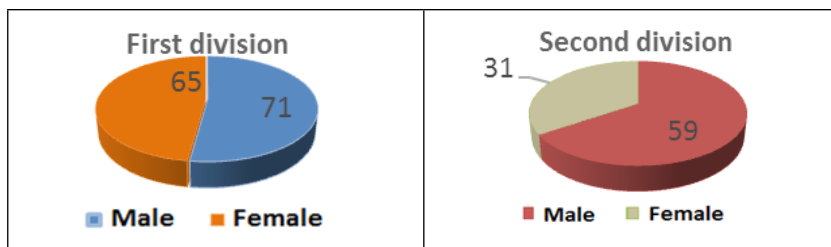


Fig. 3. Gender distribution of participants in KPM finals 2018.

5. Discussion

After two years of conducting the updated version of the marathon, it showed very important effects, the society accepted the event and adopted it very quickly, many parents were enthusiastic to send their children to this competition, as there is much more interest in computer science education in the general opinion. It is reflected this year by a very large number of participants in this year version of the marathon which is still in the first phase as this paper is prepared (about 300% increase from the last year). So, we think that the marathon is approaching its goals. We are planning to improve it and skip any problems, so we can get a pioneer experiment that can be cloned in other developing countries. We are planning to target most children in the country by cooperating with specialized organizations such as Distinction and Creativity Agency which is responsible for organizing the national informatics Olympiad, we also prepare to integrate this competition with Bebras as we are preparing to join the community shortly, the undergoing proposal is to find a plan to enroll the winners of Bebras in the marathon, to narrow and focus the efforts to train students that are really interested and have suitable talents.

The material is revised as well, we are considering using Python as a programming language instead of C++, as it seems to be more convenient for young people, and proved efficiency in many other competitions such as (ANDERLE, 2018).

We are very pleased for the wide community acceptance of the event, many institutions are willing to support the organization and scientific affairs of the marathon including Syrian virtual university, ministry of education, Distinction and creativity agency and Syrian computer society.

We think KPM starts affecting the society to push toward CSE: Many educational institutes started to organize courses to support computer science, parents are convinced to send their children to CS courses. Besides, the marathon revealed the lack of CSE in school syllabus, so that many organizations such as DCA and SCS are considering support of this type of education more seriously.

The event is promising and constitute an important factor in both improving the computer science education awareness, and support other computer science competitions that targets older people such as Informatics Olympiad and ICPC.

6. Conclusion

In this paper we presented the Kids Programming Marathon, an annual computer science competition for children aged between 8 and 15 years. We showed the motivation and goals of it, where the marathon aims at preparing new generation for the future as many jobs will be linked to computer science, the detailed information of the KPM is presented, the different phases the participants go into. We elaborated the materials used in this competition which is a combination of various types of tasks, where logical tasks enhance problem solving skills by improving the ability to connect facts logically to achieve the right solution, while solving the programming problems enhance the skills of dividing

the required task into several simpler ones and the innovation in creating appropriate solutions that meet the required target, which reflect directly on the final implementation of the program either visually with Scratch or textual with C++. We discussed some aspects of the event and showed our viewpoint for the future of it, the marathon had a positive impact on the society in the last two years, we mentioned that many organizations were inspired by the idea, many institutes started to prepare CS courses for young people and many others are considering preparing programs to fill the gap of CS in school syllabus, the kids programming marathon is a promising competition that should be supported in order to promote computer science education and for better future of our children.

References

- Alnahhas, A., Alazab, E. (2015). Selecting and training students with no suitable informatics background for informatics Olympiads – the case of Syrian Olympiad in Informatics. *Olympiads in Informatics*, 9.
- Anderle, M. (2018). *PRASK – an Algorithmic Competition for Middle Schoolers in Slovakia*.
- Hubwieser, P., Armoni, M., Giannakos, M.N. (2015). How to implement rigorous computer science education in K-12 schools? Some answers and many questions. *ACM Transactions on Computing Education (TOCE)*, 15(2), 5.
- Idlbi, A. (2009). Taking kids into programming (contests) with Scratch. *Olympiads in Informatics*, 3, 17–25.
- Resnick, M., Kafai, Y., Maeda, J. (2005). A networked, media-rich programming environment to enhance technological fluency at after-school centers in economically-disadvantaged communities.
- State of Computer Science Education*. (2018). Retrieved from <https://advocacy.code.org/>
- World Development Report 2019: The Changing Nature of Work*. (2019). Retrieved from Washington, DC: World Bank.:



M.N. Taki is a computer engineer from the Computer & Automation Engineering Department at Damascus University, and has been a scientific coordinator of Syrian Olympiad in Informatics in 2009–2011. She is the director of the Kids Programming Training program at the Syrian Virtual University. She has worked on introducing programming to the kids and youth. Her interests include promoting usage of new technologies with children and special needs and to introduce STEAM concepts in interactive learning methods.



A. Alnahhas is a teacher assistant at the faculty of information technology engineering, Damascus University, he holds a master degree in artificial intelligence and is a Ph.D. candidate, he was involved in the training and preparation of national Olympiad in informatics since 2005, he was involved in many computer science activities for children, he has been the leader of Syrian delegation to IOI for many years. He is the member of the scientific committee of Kids programming marathon since 2018.

Digital Curator

Marina S. TSVETKOVA, Vladimir M. KIRYUKHIN

Russian Academy of Natural History, Moscow, Russian Federation
e-mails: ms-tsv@mail.ru; vkiryukh@gmail.com

Abstract. This article is the announcement of a chapter “Advanced digital competence of teachers” in the new scientific publication “Teacher Education in the 21st Century” (IntechOpen, London, 2019, academic edition Reginald Monyai,). This chapter is distributed under the terms of the Creative Commons Attribution License*.

Keywords: digital economy, digital pedagogy, digital competence of teachers, digital school, digital curator.

1. Introduction

The school as a social institution has taken these digital waves upon itself, and this has influenced the extremely dynamic renewal and expansion of teachers’ competencies – from traditional to digital. This demanded from governments to pay close attention to the digital competencies of citizens, and especially teachers, who form these competencies in children not spontaneously, but systematically for the socialization of the younger generation in the information society.

A modern digital society educator must continuously enrich and complement his or her digital competence by working with the growing digital generation of aboriginal children in the digital society.

Any adult, not only a teacher, will always face a new digital wave during his life, which is generated by more and more technically advanced information and communication technologies. In this sense, an adult, and a teacher also, in the community of children, always remains an emigrant of the digital society in the new digital wave.

Now we are experiencing a digital wave of artificial intelligence – the 4th industrial revolution and the electronic economy (RDEP, 2018). It is connected with the penetration of numbers in the artificial world of things, which became possible to manage thanks to artificial intelligence already in the global information space through the Internet and mobile devices not only in the workplace but also in everyday life. In this new digital world, a teacher forms a willingness to live in a new civilization.

* <https://www.intechopen.com/books/teacher-education-in-the-21st-century>

Existing experience shows that user competencies (digital literacy) in the new conditions of the digital economy are transforming and include not only common for all user (life) digital competencies, but also professional digital competencies (profession digitalization) and new social digital competencies in the global information world. Consider this triangle of new advanced digital competencies for a teacher in the conditions of the 4th digital wave.

Cyber-worlds, in which children of our digital wave already live, are a natural environment of normal human activity, filled with virtual analogues: cyber art, cyber education, cyber offices, cyber banks, cyber police, cyber libraries, cyber enterprises, cyber medicine... All this should be included in the basic digital competence of the teacher to teach children to live in a digital world and to have an idea of the penetration of all new digital devices into it. Life digital competence of a competent user becomes a natural component of the general culture of the digital world.

The new mission of a digital pedagogy teacher is to teach children to learn in a digital environment throughout their lives. It is also important that each teacher brings to the children's community information about new professions in their subject area (ICT, 2011). Children are focused on the future. Their professional choice is formed in school, and professions are formed by the digital economy and the new digital wave also defines the digitalization of the professions.

In addition to the professions of a programmer, web designer and system administrator, which are traditionally digital for the beginning of the 21st century, the digital economy is rapidly enriching all professions with numbers and creating new professions. New professions of the near future show the dynamic addition of the surrounding world with the cyber world. Knowing about these professions, helping children to get basic professional skills with the involvement of business partners in school are the most important task for the professional choice of the child, his readiness for the challenges of the digital world.

Social digital competence of the teacher is not only psycho-oriented, but is aimed to form in children the value of education and general media literacy in working with information in the Internet and global media, prevention of cybercrime and cyber mania, fostering a culture of cyber security and Internet etiquette in global knowledge networks as opposed to entertainment. Here the teacher should act like a digital curator for the socialization of children in the digital world.

2. Digital Curator

In the new digital wave, social digital competencies require special attention for teachers to work with children. It is necessary to strengthen the environment of development of social digital competences of teachers. In many countries, teachers have already appeared-digital curators in libraries, social adaptation centers, but it is important that they are in every school.

Professional standard "Consultant in the field of digital literacy (digital curator)" is approved in Russia from 31.10.2018 by the Ministry of labor and social protection

(DC 2018). The responsible organization-developer of the professional standard was the all-Russian public and state educational organization “Knowledge” (RS, 2018).

The purpose of the new type of professional activity is to advise on the use of information and communication technologies in various spheres of life, to promote the development of digital literacy of different groups of the population.

The competencies of the digital curator are the following:

- Conducting direct reception of citizens’ appeals.
- Electronic communication on citizens’ appeals.
- Search and processing of information required for consultations in accordance with the work assignment.
- Visual and remote placement of information and consultations.
- Maintaining a database of citizens who have applied for advice.
- Explanation and demonstration of the ICT application algorithm.
- Informing about the most common threats when working in the network, using the means of communication.
- Informing about the main methods of combating cyber threats.
- Conducting surveys and questionnaires on the results of activities aimed at the development of digital literacy.
- Development of programs of information and educational activities for the development of digital literacy of various groups of citizens and the promotion of consulting services.
- Diagnostics of the level of digital literacy of the citizen who applied for consultation.
- Analysis of the market of digital products and services, digital literacy of the citizens and resources for their development (information resources, educational and enlightening programs).
- Organization of the introduction of modern methods, techniques and forms of counseling on digital literacy development, dissemination of positive experience of counseling; etc.

Digital curator should know the rules of business correspondence and written etiquette; rules of business communication and speech etiquette; requirements for documentation; norms of the native language; principles and mechanisms of search engines, functionality of popular search services. He needs to know the legislation of the country law in the field of intellectual property, personal data, types and basic user characteristics of mobile devices; basic principles of organization and functioning of computer networks. He should be familiar with the main online services for the provision of electronic services, state portals and municipal services, including services provided with the use of electronic social cards, electronic payments, electronic queues, and electronic reception. He is required to get acquainted with the trends in the development of information and communication technologies and digital literacy; the market of modern educational programs aimed at the development of digital literacy; directions and prospects for the development of ICT for the citizens; modern approaches, forms, methods and techniques of additional education and enlightenment, features of additional education and education on the development of digital literacy, etc.

Reference

- DC (2018). *Professional Standard "Consultant in the Field of Digital Literacy Development (Digital Curator)"*. http://www.consultant.ru/document/cons_doc_LAW_311506/
- ICT (2011). *The Structure of ICT Competences of Teachers / UNESCO Recommendations*. <http://unesdoc.unesco.org/images/0021/002134/213475r.pdf>
- RDEP (2018). *Russian Digital Economy Program*. <http://data-economy.ru>
- RS (2018). *Russian Society "Knowledge"*. <https://www.znanierussia.ru/useful/Pages/digital-curator.aspx>



M.S. Tsvetkova, professor of the Russian Academy of Natural Sciences, PhD in pedagogic science, prize-winner of competition “The Teacher of Year of Moscow” (1998). Since 2002 she is a member of the Central methodical commission of the Russian Olympiad in informatics, the pedagogic coach of the Russian team on the IOI. She is the author of many papers and books in Russia on the informatization of education and methods of development of talented students. Since 2013 she is the Russian team leader. Expert of Committee on Education and Science State Duma of the Russian Federation (since 2017).



V.M. Kiryukhin is professor of the Russian Academy of Natural Sciences. He is the author of many papers and books in Russia on development of Olympiad movements in informatics and preparations for the Olympiads in informatics. He is the exclusive representative who took part at all IOI from 1989 to 2017 as a member of the IOI International Committee (1989–1992, 1999–2002, 2013–2017) and as the Russian team leader (1989, 1993–1998, 2003–2012). He received the IOI Distinguished Service Award at IOI 2003, the IOI Distinguished Service Award at IOI 2008 as one of the founders of the IOI making his long term distinguished service to the IOI from 1989 to 2008 and the medal “20 Years since the First International Olympiad in Informatics” at the IOI 2009.

About Journal and Instructions to Authors

OLYMPIADS IN INFORMATICS is a peer-reviewed scholarly journal that provides an international forum for presenting research and developments in the specific scope of teaching and learning informatics through olympiads and other competitions. The journal is focused on the research and practice of professionals who are working in the field of teaching informatics to talented student. OLYMPIADS IN INFORMATICS is published annually (in the summer).

The journal consists of two sections: the main part is devoted to research papers and only original high-quality scientific papers are accepted; the second section is for countries reports on national olympiads or contests, book reviews, comments on tasks solutions and other initiatives in connection with teaching informatics in schools.

The journal is closely connected to the scientific conference annually organized during the International Olympiad in Informatics (IOI).

Abstracting/Indexing

OLYMPIADS IN INFORMATICS is abstracted/indexed by:

- Cabell Publishing
- Central and Eastern European Online Library (CEEOL)
- EBSCO
- Educational Research Abstracts (ERA)
- ERIC
- INSPEC
- SCOPUS – Elsevier Bibliographic Databases

Submission of Manuscripts

All research papers submitted for publication in this journal must contain original unpublished work and must not have been submitted for publication elsewhere. Any manuscript which does not conform to the requirements will be returned.

The journal language is English. No formal limit is placed on the length of a paper, but the editors may recommend the shortening of a long paper.

Each paper submitted for the journal should be prepared according to the following structure:

- concise and informative title
- full names and affiliations of all authors, including e-mail addresses
- informative abstract of 70–150 words

- list of relevant keywords
- full text of the paper
- list of references
- biographic information about the author(s) including photography

All illustrations should be numbered consecutively and supplied with captions. They must fit on a 124 × 194 mm sheet of paper, including the title.

The references cited in the text should be indicated in brackets:

- for one author – (Johnson, 1999)
- for two authors – (Johnson and Peterson, 2002)
- for three or more authors – (Johnson *et al.*, 2002)
- the page number can be indicated as (Hubwieser, 2001, p. 25)

The list of references should be presented at the end of the paper in alphabetic order. Papers by the same author(s) in the same year should be distinguished by the letters a, b, etc. Only Latin characters should be used in references.

Please adhere closely to the following format in the list of references:

For books:

Hubwieser, P. (2001). *Didaktik der Informatik*. Springer-Verlag, Berlin.

Schwartz, J.E., Beichner, R.J. (1999). *Essentials of Educational Technology*. Allyn and Bacon, Boston.

For contribution to collective works:

Batissta, M.T., Clements, D.H. (2000). Mathematics curriculum development as a scientific endeavor. In: Kelly, A.E., Lesh, R.A. (Eds.), *Handbook of Research Design in Mathematics and Science Education*. Lawrence Erlbaum Associates Pub., London, 737–760.

Plomp, T., Reinen, I.J. (1996). Computer literacy. In: Plomp, T., Ely, A.D. (Eds.), *International Encyclopedia for Educational Technology*. Pergamon Press, London, 626–630.

For journal papers:

McCormick, R. (1992). Curriculum development and new information technology. *Journal of Information Technology for Teacher Education*, 1(1), 23–49.

<http://rice.edn.deakin.edu.au/archives/JITTE/j113.htm>

Burton, B.A. (2010). Encouraging algorithmic thinking without a computer. *Olympiads in Informatics*, 4, 3–14.

For documents on Internet:

International Olympiads in Informatics (2008).

<http://www.IOInformatics.org/>

Hassinen, P., Elomaa, J., Ronkko, J., Halme, J., Hodju, P. (1999). *Neural Networks Tool – Nenet (Version 1.1)*.

<http://koti.mbnet.fi/~phodju/nenet/Nenet/General.html>

Authors must submit electronic versions of manuscripts in PDF to the editors. The manuscripts should conform all the requirements above.

If a paper is accepted for publication, the authors will be asked for a computer-processed text of the final version of the paper, supplemented with illustrations and tables, prepared as a Microsoft Word or LaTeX document. The illustrations are to be presented in TIF, WMF, BMP, PCX or PNG formats (the resolution of point graphics pictures is 300 dots per inch).

Contacts for communication

Valentina Dagiene
Vilnius University
Akademijos str. 4, LT-08663 Vilnius, Lithuania
Phone: +370 5 2109 732
Fax: +370 52 729 209
E-mail: valentina.dagiene@mif.vu.lt

Internet Address

All the information about the journal can be found at:

<https://ioinformatics.org/page/ioi-journal>

Publisher office: Vilnius University
Akademijos str. 4, LT-08663 Vilnius, Lithuania
August, 2019

Olympiads in Informatics

Volume 13, 2019

S. COMBÉFIS, G. DE MOFFARTS, M. JOVANOVIĆ	3
TLCS: A Digital Library with Resources to Teach and Learn Computer Science	
M. DOLINSKY, M. DOLINSKAYA	21
Training in Writing the Simplest Programs from Early Ages	
D. GINAT	31
On Implicit Means of Algorithmic Problem Solving	
M. JANCHESKI, S. JANCHESKA	41
Multidisciplinary, Multilingual, Multilevel and Multipurpose Usage of GeoGebra Software in Education	
T. KAKESHITA, M. OHTSUKI	57
Survey and Analysis of Computing Education at Japanese Universities: Non-IT Departments and Courses	
T. KAKESHITA, N. TAKAHASHI, M. OHTSUKI	81
Survey and Analysis of Computing Education at Japanese Universities: Informatics in General Education	
M. LODI, D. MALCHIODI, M. MONGA, A. MORPURGO, B. SPIELER	99
Constructionist Attempts at Supporting the Learning of Computer Programming: A Survey	
K. SUMI, M. OHTSUKI, T. KAKESHITA	123
Survey and Analysis of Computing Education at Japanese Universities: Subject of “Information” for High School Teacher’s License	
W. van der VEGT, E. SCHRIJVERS	145
Analyzing Task Difficulty in a Bebras Contest Using Cuttle	
T. VERHOEFF	157
Programming, Software Development, and Computer Science – The Golden Triangle	
M. WEIGEND, J. VANÍČEK, Z. PLUHÁR, I. PESEK	171
Computational Thinking Education Through Creative Unplugged Activities	
REPORTS	
P. ERACLEOUS, P. PAVLIKAS, A. TTOFARI, A. CHARALAMPOUS. Cyprus Olympiad in Informatics	193
M. MEDVEDIEV. The Use of E-Olymp Internet Portal in Programming Competitions	201
K. MIRJALALI, A. Keivan MOHTASHAMI, M. ROGHANI, H. ZARRABI-ZADEH. TPS (Task Preparation System): A Tool for Developing Tasks in Programming Contests	209
Y. TABESH, S. ZARKESH, A. ZARKESH, I. FAZILOVA. Computational Thinking in K-12: Azerbaijan’s Experience	217
M. TAKI, A. ALNAHHAS. Kids Programming Marathon: A Step toward Better Engagement with Computer Science Education	225
M.S. TSVETKOVA, V.M. KIRYUKHIN. Digital Curator	237

Olympiads in Informatics

Volume 13, 2019

S. COMBÉFIS, G. DE MOFFARTS, M. JOVANOVIĆ TLCS: A Digital Library with Resources to Teach and Learn Computer Science	3
M. DOLINSKY, M. DOLINSKAYA Training in Writing the Simplest Programs from Early Ages	21
D. GINAT On Implicit Means of Algorithmic Problem Solving	31
M. JANCHESKI, S. JANCHESKA Multidisciplinary, Multilingual, Multilevel and Multipurpose Usage of GeoGebra Software in Education	41
T. KAKESHITA, M. OHTSUKI Survey and Analysis of Computing Education at Japanese Universities: Non-IT Departments and Courses	57
T. KAKESHITA, N. TAKAHASHI, M. OHTSUKI Survey and Analysis of Computing Education at Japanese Universities: Informatics in General Education	81
M. LODI, D. MALCHIODI, M. MONGA, A. MORPURGO, B. SPIELER Constructionist Attempts at Supporting the Learning of Computer Programming: A Survey	99
K. SUMI, M. OHTSUKI, T. KAKESHITA Survey and Analysis of Computing Education at Japanese Universities: Subject of “Information” for High School Teacher’s License	123
W. van der VEGT, E. SCHRIJVERS Analyzing Task Difficulty in a Bebras Contest Using Cuttle	145
T. VERHOEFF Programming, Software Development, and Computer Science – The Golden Triangle	157
M. WEIGEND, J. VANÍČEK, Z. PLUHÁR, I. PESEK Computational Thinking Education Through Creative Unplugged Activities	171
REPORTS	
P. ERACLEOUS, P. PAVLIKAS, A. TTOFARI, A. CHARALAMPOUS. Cyprus Olympiad in Informatics	193
M. MEDVEDIEV. The Use of E-Olymp Internet Portal in Programming Competitions	201
K. MIRJALALI, A. Keivan MOHTASHAMI, M. ROGHANI, H. ZARRABI-ZADEH. TPS (Task Preparation System): A Tool for Developing Tasks in Programming Contests	209
Y. TABESH, S. ZARKESH, A. ZARKESH, I. FAZILOVA. Computational Thinking in K-12: Azerbaijan’s Experience	217
M. TAKI, A. ALNAHHAS. Kids Programming Marathon: A Step toward Better Engagement with Computer Science Education	225
M.S. TSVETKOVA, V.M. KIRYUKHIN. Digital Curator	237