Computer Science and Information Systems 15(3):775–798 https://doi.org/10.2298/CSIS180312028K

Using Screencasts to Enhance Coding Skills: the case of Logic Programming*

Petros Kefalas and Ioanna Stamatopoulou

The University of Sheffield International Faculty, CITY College
Computer Science Department
3 Leontos Sofou Street
Thessaloniki, 54626
Greece
{kefalas, istamatopoulou}@citycollege.sheffield.eu

Abstract. Learning technologies are gradually becoming an integral part of teaching in both face-to-face and online learning. Among them, screencasts (i.e. desktop video recordings of presentations normally accompanied by the presenter's video and narration), constitute a pedagogical tool used to create visual material to be distributed to students. Learners can then watch the videos in their own convenience and pace. The plethora of tools available makes it easier for the teachers to produce high-quality, low-cost screencasts for a number of courses. In the current paper we investigate how students perceive the impact of screencasts on their understanding and motivation in learning Logic Programming. We take the opportunity to present some tips and techniques that can be applied in any screencast production. We discuss in detail how screencasts can be used in programming courses, irrespectively of whether the latter use Imperative, Object-Oriented, or Declarative languages, and we present a number of examples to demonstrate how screencasts can facilitate learning. Furthermore, we focus particularly on Logic Programming, which lends itself to technology enhanced learning, since it requires a non-linear, out-of-the-box way of thinking towards developing programs. Finally, we evaluate our approach by presenting the opinion of students.

Keywords: Screencasts, Learning Technologies, Logic Programming.

1. Introduction

Learning Technologies (also often referred to as educational technologies) are considered as the methods and tools that use information and communication technologies (ICT) as a means to enhance learning through teaching, assessment and feedback. The basic idea is that teachers use computer-based applications integrated into any facet of education provision with the purpose of engaging students more than a traditional setting would, and thus enhance their learning experience.

The growth of learning technologies follows the development of the ICT. On one hand, network communication allows live multi-media streaming, and recorded video and sound storage and retrieval in a more efficient manner than a few years ago. On the other

* Extended version of Kefalas, P., Stamatopoulou, I. (2017) Using Screencasts to Enhance Logic Programming Skills. In: Proceedings of the 8th Balkan Conference in Informatics (BCI2017), 20-23 September 2017, Skopje. ACM, New York, NY, USA.

hand, the affordable prices of devices connected to the Internet, or other instructional devices, have made their use and adoption for educational purposes easy. Finally, a plethora of desktop, mobile, and collaboration tools has emerged with the aim of simplifying the "home-production" of educational material and applications, as well as to facilitate interaction between teacher and learners, and among learners themselves. All the above were literally just a wish list a decade ago.

The vast majority of learning technologies facilitates what we normally call e-learning or online learning. However, it is often the case that practices from e-learning courses can be transferred immediately to face-to-face teaching and therefore improve traditional teaching in many different ways. The new, blended learning mode, offers the best practices of both face-to-face and online learning.

One of the learning technologies that is being increasingly utilised lately is video recordings made by teachers and lecturers. A screencast (also known as video screen captures) is a digital recording, of a whole or part of a computer screen, that is often accompanied by audio narration. Screencasts are considered as autonomous learning objects and are useful for a number of educational purposes; educators can record a lecture, a "how-to" tutorial on various tasks, give feedback to students, compile a library of frequently asked questions and corresponding answers, etc.

There exists an abundance of simple-to-use tools that record the activity on a computer screen and also provide simple interfaces for supporting tasks, such as editing. Lately, such tools also offer a wide variety of options with regard to the video format type and archiving, and thus they facilitate the distribution of these learning objects. It is important that these tools, and their associated functionalities, have made the recording and distribution of "home-made" desktop videos much more effective, requiring no special equipment or setup.

This work continues our previous work [13] by expanding the literaure review, incorporating the questionnaire results of one additional cohort, and arguing that our approach can be further generalised to support the learning of other programming languages, such as Java. The paper is organised as follows: Section 2 explicitly states this work's contribution, our research question, and our methodology towards addressing it. Section 3 presents a survey of existing related literature and Section 4 introduces the way screencasts can be used in higher education and the practices we follow in our faculty. In Section 5, we discuss a number of tips for creating screencasts while Section 6 discusses the advantages, for students, of developing screencasts solving simple programming exercises. The specific case study for Logic Programming is presented in Section 7, and the results of our students' survey in Section 8. In Section 9 we briefly discuss the human factor in employing learning technologies and screencasts in particular. Finally, Section 10 concludes the paper and discusses some further work.

2. Contribution and Methodology

In this paper, we address educators in Computer Science programmes in order to encourage the production of screencasts for programming exercises. Object-Oriented programming (e.g. in Java) as well as Declarative programming (e.g. in Prolog) are discussed with respect to their suitability for developing screencasts with coding examples, with more focus placed on Prolog, a logic programming paradigm for which students are required to think in a rather different (declarative) way than the one they are accustomed to (procedural) in object-oriented or imperative languages. Our aim is to share our experience and practices with colleagues who would like to employ screencasts as a technology enhanced learning (TEL) method.

Our contributions are the following: (a) we present some ideas on how screencasts can be utilised in Higher Education programmes, (b) we offer a set of guidelines and tips on how to develop "home-made" screencasts that are of good technical quality, and (c) we address the research question of *how students perceive the impact of screencasts on their understanding and motivation in learning Logic Programming*, aiming to demonstrate the effectiveness of screencasts in supplementing teaching programming languages, and in particular Prolog.

To verify whether screencasts have a positive effect on students' understanding of Logic Programming and on their motivation in learning it, we decided to distribute a questionnaire. The short survey included the following questions:

- What has been the impact of the screencasts on your understanding of Logic Programming?
- What has been the impact of the screencasts on your motivation to learn/practice?
- What did you like most?
- What would you like to change?
- Would you like to have screencasts in other courses? If yes, in which course(s) and in what context?

For the first two questions, which directly address our research question, students were required to respond by selecting one of the following six options: Strongly positive, Positive, Neither positive or negative, Negative, Strongly negative, Have not seen any of the screencasts. The next two questionnaire questions were open ended. The final was a yes/no question allowing students to clarify in open-ended form in the case their response was yes. The participants have been all students who attended the course during the past three years (due to the relatively small size of our department no sampling was considered necessary), i.e. a total of 57 students. The questionnaire was created online (using Google forms) and anonymous.

3. Review on Using Videos in Learning

In the literature one can find many attempts to incorporate video in teaching and learning. Especially during the last decade, video has become a medium that is easily accessible through internet and mobile technology while students seem to watch videos more than reading texts for various reasons, including learning. In particular, screencasts have been reported as an effective tool for student learning in general [9], [17], [23] or in specific tasks, such as programming [19].

Videos in the form of screencasts are considered better than texts [6], [5], [21]. Students prefer not to read instructions or manuals but rather to watch videos in a multimedia blended learning environment [28] or in flipped classrooms [32], [18]. Although there is not as much research on the pedagogical effectiveness of videos as one should expect after the plethora of MOOCs available [12], there is some evidence that students who engage in watching videos, such as screencasts or lecture captures, perform significantly better than those who do not [34], [21], [22]. Videos have been shown to attract student interest more and encourage debates in classrooms [31]. As a more general rule students seem to enjoy blended learning techniques more because of immediacy and currency [21]. Other factors, such as personal learning styles, study habits, options for interaction, etc. play an important role to whether students use videos in their studies [35]. Moreover, different video types seem to affect learning in various ways, most favourable being lecture captures and screencast type videos [3].

In programming courses, screencasts are even more effective, especially when they complement face-to-face teaching [26]. Students tend to assimilate the taught material much better and therefore acquire better programming skills [7], which eases the transition to gradual program development [15]. Especially, if learning programming is considered equivalent to learning a second language (syntax, semantics, processes, etc), short videos can facilitate learning as analogous to language acquisition [30]. In some cases, students have been asked to produced their own screencasts in programming thus also improving their analytical skills [20].

Screencasts are more effective if they are produced by lecturers who are appropriately trained to develop high-quality videos. There is a consensus that faculty should be trained in order to produce screencasts that follow the basic principles and pedagogies required in order to enhance student learning [6]. Universities need to confront the faculty unwillingness, inability or lack of interest in order to change the traditional textbook-related pedagogy [10]. Although the creation of screencasts is considered time consuming it is considered to be definitely worth investing on [26]. The goal is to introduce the faculty not only to the technical aspects of creating videos but also to the ways of maximising student engagement with these videos, by following recommendations related to duration, learning goals, signaling, teaching and guiding style, etc. [2], [10], [12]. Finally, learning analytics could further enhance the development and tactics of videos production by focusing on aspects of usability and acceptance by students [8]. It is also suggested that an integrated environment can be used in order to make videos more interactive [29], [31].

4. The use of Screencasts

In our Faculty, TEL is highly promoted as a University strategic goal, called “digital learning”.[1] All academic staff is trained, through staff development seminars and personal tutorials, to use various learning technologies and tools that facilitate mixed-mode delivery approaches. This is particularly useful for programmes that our faculty runs abroad as distributed learning programmes [14], in which students receive face-to-face instruction once a month over “block teaching” but are also engaged in structured e-leaning activities in-between physical attendance sessions. Given this experience, staff also uses learning technologies to complement traditional teaching and learning, and, among them, screencasts are used in a variety of ways. In this section, we present some of them (Fig. 1).

Recorded Lectures The most obvious reason to use screencasts is to record (capture) lectures in “live” mode while they are actually taking place, a feature that has been adopted to

[1] https://www.sheffield.ac.uk/als/strategy

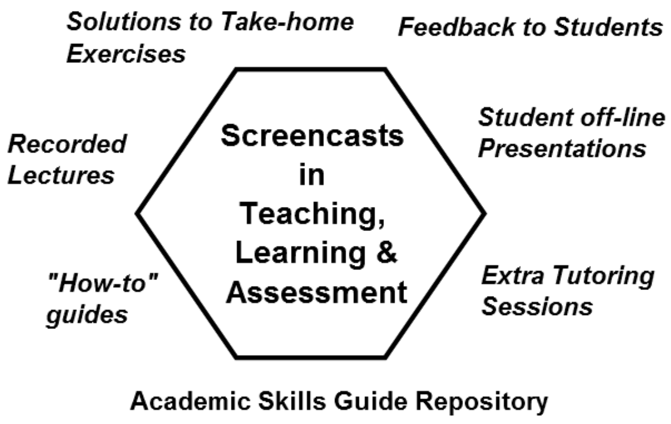

Solutions to Take-home Exercises
Feedback to Students
Recorded Lectures
Screencasts in Teaching, Learning & Assessment
Student off-line Presentations
"How-to" guides
Extra Tutoring Sessions
Academic Skills Guide Repository

Fig. 1. Teaching, Learning and Assessment activities that can employ Screencast recordings

complement traditional lectures in many universities but is mostly used in e-learning programmes and MOOCs. Our University has established a lecture capture policy, by which lectures in the near future will be by default recorded and made available to students on the virtual learning environment (VLE).

While lecture capturing is considered as highly desirable by students, it has raised a number of controversies among academics, the majority of whom are opposed to it [25]. The main arguments include possible lack of interactivity, concerns about increased absenteeism, copyright issues, confidentiality, etc.

Recording is also done on various occasions in “off-line” mode in the creators’ own convenience at the desktop of their computers. Instructors record their presentations so that students have the ability to watch them at a later stage. This practice is commonly found in flipped classroom settings, whereby recorded material is provided to the students before the class takes place, and face-to-face instruction focuses only on discussing the material seen in advance.

We also use desktop capturing quite often as brief introductions to courses. We call these videos “teasers” and we distribute them to students before the commencement of a semester. A teaser is a short recording that explains what a course is all about in a rather informal way, outlines a course’s aims and learning outcomes but most importantly answers the questions why students should attend it and what to expect from it.

On other occasions, we also broadcast and record lectures of guest speakers in open seminars, and these recordings are made available to lecturers for future use in their courses. Of course, permission to do so is taken from the guest speaker, ensuring also that the material presented (presentation slides most often) does not contain anything copyrighted.

Solutions to Exercises A number of our colleagues have used screencasts to show students how to solve exercises, i.e. to demonstrate what we normally call a “model answer” to a particular question, especially in courses such as programming, maths, accounting, etc. This method is seen positively both by students and academic staff, who develop confidence that such a repository helps weaker students as they have the opportunity to go through a solution many times while listening to the lecturer explain how the solution is reached. In principle, such screencasts seem to have an added value when there is some underlying methodology involved in answering a question or solving an exercise, showing a step-by-step approach that can be followed to address a category of similar problems.

Feedback to Students Among other digital methods for providing feedback, we also utilise screencasts to address the whole class. After individual feedback is given, the lecturer prepares a short screencast to present common pitfalls gathered from all assignment submissions together with performance statistics. The screencast is distributed to all students who have the opportunity to watch it on their own time and understand what were the common mistakes as well as how they did in comparison to others.

In some cases screencasts are also used to provide individual feedback to students. Thus, the lecturer individually addresses each student explaining what they have done well (or nor so well), very likely also including comments on their submission, which appear on the screen as a document or listing. This practice is particularly useful for students as it helps them understand how the lecturer approaches assessment using specific criteria and how their overall mark is allocated.

Student Presentations During their studies, our students deliver a number of presentations to live internal or external audiences. It has been a few years now that we started asking for screencasts of some of these student presentations, as we strongly believe that students should acquire online presentation skills that will help them in their professional career. To prepare them we organise specific training sessions to demonstrate the methodology and the tools and then ask them to prepare a 5–8-minute screencast presentation as a coursework assignment. Students upload their presentations on the cloud and submit the link, and, besides the content-related assessment criteria, we also assess them with respect to technical issues relating to the screencast and their online presentation skills. Students report that they find this task an interesting experience that makes them think of different approaches to presenting and gives them some base-line essential skills.

Academic Skills Development Repository In the context of developing academic and transferable skills, students are required to learn how to read, study, prepare for examinations, cite and reference, avoid plagiarism, manage their time, write an academic report, etc. We have developed a screencast repository that addresses all the above issues and more. Therefore, the academic staff often direct students to this site for them to improve the skills required or to understand how to address difficulties they face, especially at the beginning of their university studies. This repository is also very useful when proving feedback on assignments, since the lecturer does not only point out the issues that students face but can direct them to the corresponding screencasts for revision and practice.

5. Tips for creating Screencasts

There is a plethora of sites and books that provide guidance on how to develop virtual presentations [33]. In [11], the authors provide a set of recommendations for the production of videos in order to keep students more engaged. “Home-production” video is a contemporary skill that academic staff as well as students should develop while living in the digital era. In this section we present a number of tips that can dramatically affect the quality of a screencast for the better; these are also used as criteria to assess the student recorded presentations mentioned in the previous section. We assume that the computer (PC or laptop) used for their creation has a camera, a microphone or headset, and is optionally connected to the Internet.

Background People tend to neglect what is behind them as a background image when recording themselves in video. Keeping it simple and neutral while contrasting with the image of the speaker is a key feature of appropriate background. We found that a poster or a banner behind the speaker can convey useful messages, for instance about the screencast topic or the programme, the University, etc.

Environment setup The two main issues related to the environment are light and sound. If the video of the presenter is on the screencast then good lighting is important so that there are no distorting shadows on the image. Background noise may also pose a problem, since the human ear can filter it out better than a microphone; as a result the noise is magnified to the ears of the viewer. The use of a headset can reduce the noise and clarify the presenter’s voice but affects the image of the presenter on the video, and thus it is a matter of compromise. It is also important to secure the environment from any disruptions, such as telephone rings or door knocking, so that repetitions and editing are minimised.

Camera positioning The camera should be positioned at the level of the presenter’s face, in a way that the viewers perceive a direct eye contact. Sometimes the positioning of a fixed laptop camera captures the presenter from an angle that creates distortions, shadows and uneven lighting on the presenter’s face. Similar unwanted effects are caused by the distance of the camera; positioning it too far would imply a wide background image whereas positioning it too close could result in distortion of facial characteristics.

Microphone The microphone is perhaps even more important than the camera, since a screencast may not include a video of the presenter but it can rarely be without their voice. Good quality sound is important and so it is worth investing in a microphone that filters surrounding noises as much as possible. If the presenter does not want to capture themselves on video, a headset is preferable. In such a case, the distance of the microphone from the face is very important as it can cause unwanted sound distortion due to breathing.

Desktop setup Since a screencast contains whatever is shown on the computer screen, it is desirable to close all unnecessary running programmes, particularly those that might display pop-up notifications on the screen, such as email clients, skype, dropbox, etc. In the opposite case, the notification, which will be recorded in the video, will result in additional editing or a repetition of the recording. In cases where switching between screens is necessary, it is desirable to avoid showing the desktop contents or other folders’ list of files.

Script Generating a script is a good practice, especially for inexperienced presenters or novice screencast developers. The script could be as abstract or analytical as desired depending on the presenter's needs. The main advantage of having a script in a screencast as opposed to live presentation is that the script may be hidden behind the camera and not be visible to the audience, thus creating an "autocue" effect.

Choice of tool It is important to choose the right tool that fits the purpose of the screencast and the needs of the creator. The factors that play a role in the decision are primarily ease-of-use, cost, variety in file formats, editing functionalities, and archiving. Screen capturing tools may be either desktop or web applications and they normally come with limited capabilities for free or with full-blown functionality for some fee. Indicative examples of such tools include Screencast-O-Matic[2], Jing[3], EzVid[4], Camtasia[5] and many more.

Editing Editing comes as an option depending on the choice of tool and whether subscription is required. However, editing seems to be an important feature that could potentially save a lot of time in the video development. Without editing, even the smallest "errors" could trigger repetition of the recording and that could be extremely annoying if repeated few times. With editing, all "errors" are cut while certain parts of the video are adjusted to look better to students.

Archiving It is generally considered far better to be able to store the screencast on the cloud so that students can access it through a link. As a result, the choice of vendor is important as it may affect the life-span of the screencast. In any case, a local backup will solve any potential issues that may appear in the future with the chosen service provider.

No apparent audience It takes some time to become familiarised with talking to no apparent audience, due to the lack of interaction, and people tend to attempt to record an entire screencast in one (usually "breathless") go. Frequent pauses are necessary; a pause of a few seconds is perceived differently by the presenter than by the audience, and it is generally considered a good practice. Additionally, proper use of the pause button can save a considerable amount of editing later on.

6. Screencasts for Programming

Programming is considered one of the essential skills to be acquired by students in Computer Science programmes in Higher Education. It is a skill that must be included in all Computer Science or relevant programmes [1], and, what is more, students are exposed to different types of programming, becoming familiar with Object-Oriented, Imperative, and Declarative languages. Instances of such languages, such as Java, C, Prolog, etc., are adopted to show students the major paradigms with which they will acquire the basic skill

[2] screencast-o-matic.com
[3] www.techsmith.com/jing.html
[4] www.ezvid.com
[5] www.techsmith.com/camtasia.html

set that will help them code in any other programming language of the same paradigm, passing through a steep learning curve.

Researchers have attempted to create videos in programming courses, such as Prolog language, C#, etc. [4],[27]. Students who attended classes found videos equally important to students who do not and seem to prefer videos over a classic textbook [27]. In addition, it is demonstrated that student engagement increases, especially when screencasts are associated with in-video quiz questions [4]. The outcome comes into alignment with [24] who showed that interactive dynamic visualisations allow learners to acquire the necessary skills in less time than those who do not interact.

Students, as novice programmers, face the following challenges:

- understanding of syntax and semantics;
- effective and correct utilisation of the variety of data types and data structures;
- comprehension of the problem to solve;
- choosing among a variety of techniques and algorithms that leads towards a solution; and
- engineering of the code (design, implementation, and testing) that solves the problem.

The latter includes the essence of programming skills, i.e. students need to assemble a finite set of tools (commands, techniques, algorithms, designs, etc.) complying to a very strict set of rules that defines the language's syntax and semantics.

The traditional way to teach programming is by using either a whiteboard and/or a set of presentation slides. The former is preferred by many instructors who have the opportunity to develop programs in a step-by-step manner for a specific example, talk through it, and justify the decisions made at each point. The latter is also useful, particularly if animation is utilised. The drawback of the whiteboard is that the handwriting may sometimes be illegible while there is no electronic record to be played back. An interactive board can be advantageous in that respect as it can facilitate recording and replaying the steps when students go through the material while studying. However both electronic board notes and slide presentations lack narration.

The situation becomes more ineffective when a considerable number of take-home exercises are assigned. Normally, there is no time to go through each one of them and present the model solutions. Instead, a set of solutions with the actual required code is given to students. Although these solutions have some value with respect to understanding code, they have little contribution to actual code developing skills.

For instance, consider a Java take-home exercise that requires students to define a method that finds and returns the maximum number inside an integer array.

The code distributed to students as a model answer is:

```
public static int maximum(int[] numbers){
    int max = numbers[0];
    for(int i = 1; i < numbers.length; i++){
        if(numbers[i] > max)
            max = numbers[i];
    }
    return max;
}
```

While students find the above code useful (especially those who have attempted the exercise up to a point), it can have a bigger impact on their learning if it were accompanied by a step-by-step guide on how it was developed. The particular example, as any exercise requiring the definition of a Java method, could be used to teach students what to consider when it comes to defining *any* method before considering the particular problem at hand, as well to teach them how particular techniques (such as iterating over the elements of an array) are reused to solve similar problems.

Going through the solution of the above (or any similar) exercise can help students learn that, to start with, for the definition of any method, the first thing to consider is its signature, even if we are not yet certain about its return type and its parameters, making sure we include a pair of parentheses after the method name and the pair of curly brackets that defines its body, so as to avoid these accidental but common syntax errors.

```
public static <???> maximum(<???>){
    ...
}
```

The parameters are the data that the method needs in order to perform its task. For a method to be able to find the maximum within *any* array of integer numbers, it needs to be provided with an array of integer numbers. To identify the return type we need to consider what is the type of the result we are asking the method to come up with. In this case it is one integer number, the maximum.

```
public static int maximum(int[] numbers){
    ...
}
```

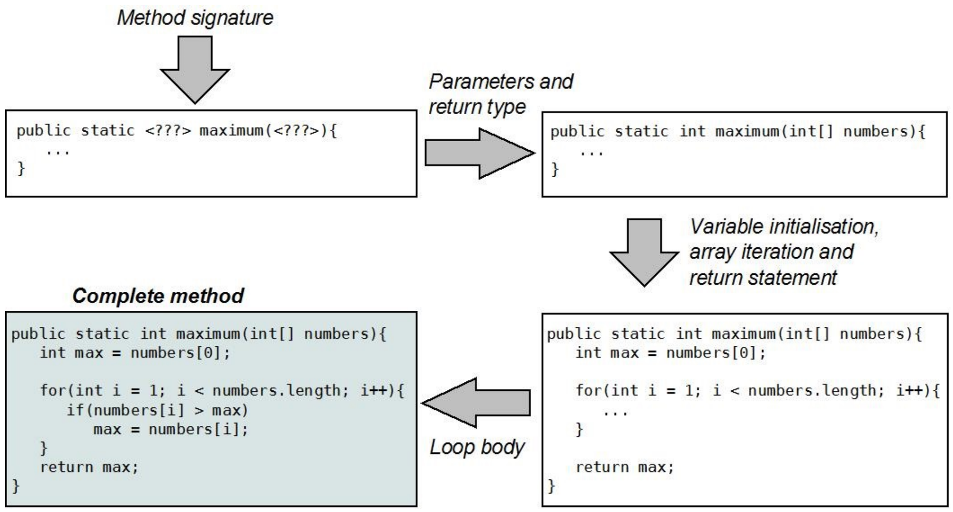
When the task of a method is to identify and return a particular element within an array, the algorithmic technique used in the imperative programming paradigm requires that (a) we assume that the element we are looking for is the first one in the array (position 0) and store it in an appropriate result variable, (b) we go through the rest elements of the array with the use of a `for` repetition structure looking for the required element, and (c) at the end we return the value of the result variable.

```
public static int maximum(int[] numbers){
    int max = numbers[0];

    for(int i = 1; i < numbers.length; i++){
        ...
    }

    return max;
}
```

This technique implies that the result variable, at any point, stores the maximum number found so far while searching inside the array. The repetition structure is used to compare each of the remaining array elements with the maximum found so far; if an array element is actually bigger than the result found so far, the latter is updated to store the value of the former.

```
public static int maximum(int[] numbers){
   int max = numbers[0];
   for(int i = 1; i < numbers.length; i++){
      if(numbers[i] > max)
         max = numbers[i];
   }
   return max;
}
```

The above process can be summarised in Fig. 2.


Method signature
Parameters and return type
Variable initialisation, array iteration and return statement
Loop body
Complete method

```
public static <???> maximum(<???>){
   ...
}
```

```
public static int maximum(int[] numbers){
   ...
}
```

```
public static int maximum(int[] numbers){
   int max = numbers[0];

   for(int i = 1; i < numbers.length; i++){
      ...
   }

   return max;
}
```

```
public static int maximum(int[] numbers){
   int max = numbers[0];

   for(int i = 1; i < numbers.length; i++){
      if(numbers[i] > max)
         max = numbers[i];
   }
   return max;
}
```

Fig. 2. Diagrammatic incremental development of a Java program that finds the maximum in an array of integers

Naturally, a number of similar exercises are presented and solved throughout the duration of a Java programming course. Students, however, find it difficult to remember (a) that the first two steps of the process, as it was presented above, should be followed when defining *any* method, and (b) that the later steps constitute a technique that can be generalised to develop any method whose task is to identify a particular element within an array of any type. The use of screencasts, for narrating and demonstrating the above for multiple exercises, helps students identify this pattern, through repetition.

7. Case: Screencasts for Logic Programming

7.1. Prolog

Logic Programming is a programming paradigm that differs from what Computer Science students consider as the "norm", which is usually imperative or object-oriented programming languages (e.g. C, C++, Java, etc.). It is included in the Computer Science curriculum with the aim of introducing students to the declarative programming paradigm, help them understand in what ways it significantly differs from the imperative one, and demonstrate the advantages and disadvantages of logic versus imperative programming. Prolog is most commonly used as a representative language since it is the most practical and widely adopted realisation of the logic programming paradigm.

Declarative programming as well as other courses, such as Discrete Maths, Formal Methods, etc., are based on the principle that one needs to describe "what the problem is" rather than "how to solve it". The major advantage of such a computational model is its simplicity in syntax and semantics, which combined with the limited number of available data types and data structures, gives the opportunity to teachers to focus on programming methodology, parameter passing, recursion, modular design and test, etc.

A Logic Program is a set of Horn clauses of the form: $H \leftarrow B_1 \wedge B_2 \wedge ... \wedge B_n$ where H is the head of the clause and $B_1 \wedge B_2 \wedge ... \wedge B_n$ is the body of the clause. The semantics of the clause is that H is true if $B_1 \wedge B_2 \wedge ... \wedge B_n$ is true. The procedural semantics is that in order to prove that H is true one needs to prove (irrespectively of order) that $B_1, B_2, \dots B_n$ are true.

A disjunction is expressed with the use of multiple clauses, such as:
$H \leftarrow B_{1_1} \wedge B_{1_2} \wedge ... \wedge B_{1_n}$
...
$H \leftarrow B_{m_1} \wedge B_{m_2} \wedge ... \wedge B_{m_k}$

which expresses that H can be proven to be true (irrespectively of order) by any of the m different clauses.

Prolog (*Pro*gramming in *Log*ic) is the most representative paradigm of the class of Logic Programming languages. For example, in Prolog, the predicate declaring the maximum of two numbers is expressed as follows:

```
max(X,Y,X):- X >= Y.
max(X,Y,Y):- X < Y.
```

meaning that the maximum between two numbers X and Y (first two parameters) is X (third parameter) if X is greater or equal to Y, or the maximum of two numbers X and Y is Y if X is less than Y.

Unlike imperative languages, in which a programme is a sequence of commands that instruct the computer how to perform a specific algorithmic task, a logic programme is a set of statements (predicates) that defines *what* the problem is and *not how* to solve it, leaving the way of solving the problem to the machine itself. This requires an "out-of-the-box" way of approaching a programming task, which, most of the times, students find quite difficult to acquire.

In the following example, we demonstrate how one should go about developing a Prolog program that finds the maximum number out of a list of integers. Lists in Prolog are sequences of elements enclosed in square brackets, e.g. $[3, 5, 1, 4, 8, 2]$, and in their general form are written as [H|T], where H is the first element of the list (Head) and T is the list that contains the rest of the elements (Tail). The list is a recursive structure and, as such, in most cases it is manipulated using recursion as a programming methodology.

As an example, consider the predicate `max/2` (max is the name of the predicate and 2 is the number of arguments/parameters that it takes), which can be used to find the maximum of a list of integers:

```
?- max([3, 5, 1, 4 , 8, 2], Max).
Max = 8
```

Considering that H represents the first element of a list, that the tail is the list containing all remaining elements, and according to the principle of recursion: *"If I know the maximum M in a list's tail, then the maximum in the whole list is either M, if H is less than M, or H, if H is greater or equal to M. In the simplest case of a list with only one element [X], I know for certain that the maximum is the element X"*.

Building the Prolog programme that corresponds to the above definition is not writing a sequence of commands. It requires thinking about the problem recursively and then trying to frame the recursive definition as a set of statements in the appropriate syntax.

Firstly, one should write the fairly standard partial recursive definition:

```
max([H | T], Max) :-
    max(T, M), ...
```

Since the maximum depends on the value of the head H of the list and there are two cases for it, one can just copy and paste the above partial programme twice:

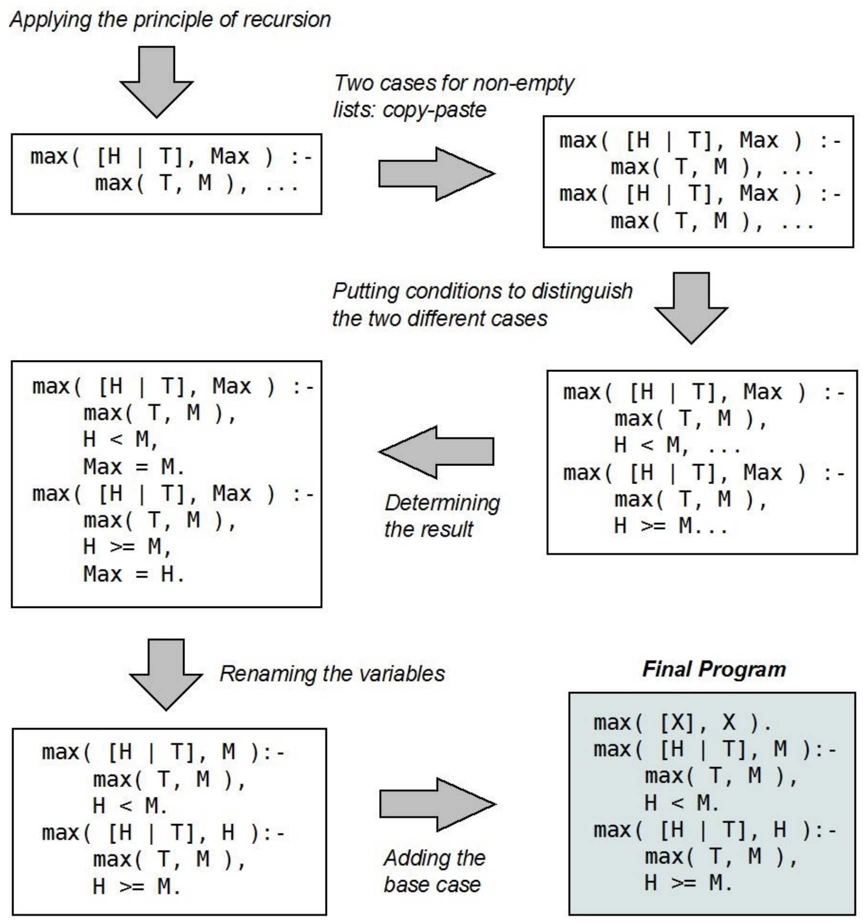
```
max([H | T], Max) :-
    max(T, M), ...
max([H | T], Max) :-
    max(T, M), ...
```

and then add the discriminating condition for each case, i.e. either $H<M$ or $H \geq M$:

```
max([H | T], Max) :-
    max(T, M),
    H < M, ...
max([H | T], Max) :-
    max(T, M),
    H >= M...
```

In the first case the maximum is M, whereas in the second case the maximum is H:

```
max([H | T], Max) :-
    max(T, M),
    H < M,
    Max = M.
max([H | T], Max) :-
    max(T, M),
    H >= M,
    Max = H.
```



Applying the principle of recursion
Two cases for non-empty lists: copy-paste
Putting conditions to distinguish the two different cases
Determining the result
Renaming the variables
Adding the base case
Final Program

```
max( [H | T], Max ) :-
    max( T, M ), ...
```

```
max( [H | T], Max ) :-
    max( T, M ), ...
max( [H | T], Max ) :-
    max( T, M ), ...
```

```
max( [H | T], Max ) :-
    max( T, M ),
    H < M, ...
max( [H | T], Max ) :-
    max( T, M ),
    H >= M...
```

```
max( [H | T], Max ) :-
    max( T, M ),
    H < M,
    Max = M.
max( [H | T], Max ) :-
    max( T, M ),
    H >= M,
    Max = H.
```

```
max( [H | T], M ):-
    max( T, M ),
    H < M.
max( [H | T], H ):-
    max( T, M ),
    H >= M.
```

```
max( [X], X ).
max( [H | T], M ):-
    max( T, M ),
    H < M.
max( [H | T], H ):-
    max( T, M ),
    H >= M.
```

Fig. 3. Diagrammatic incremental development of a Prolog program that defines the maximum of a list of numbers

By adding the termination condition (base case), and eliminating the use of "=" (since Max is either M or H we can avoid stating that they are the same, and simply replace Max with M or H, accordingly), the complete definition becomes:

```
max([X], X).
max([H | T], M):-
   max(T, M),
   H < M.
max([H | T], H):-
   max(T, M),
   H >= M.
```

It is evident that this process is not linear in the sense of imperative programming, i.e. start at the first line and write code that performs the task.

The above process can be summarised with the use of a diagram, as shown in Fig. 3. However, for bigger programs a static representation like this, or even a slide animation, becomes unmanageable. We suggest alternatively that this different way of thinking can be demonstrated to students with the aid of a screencast.

7.2. Screencasts for Prolog

The teaching and learning methods for our Logic Programming course consist of a series of interactive lectures, ten hands-on laboratory sessions, multiple-choice questions for self-assessment before each lab session, and screencast solutions to exercises for independent learning and reflection. By the end of the course, students are expected to be able to:

- understand the fundamentals of the Prolog language, such as term unification, execution, the backtracking mechanism, and the means for modifying Prolog's default behaviour;
- identify the need for recursion and apply it in solutions implemented in Prolog;
- explain the use of Prolog's built-in and higher-order predicates and employ them in their problem solving;
- determine and explain the functionality of given Prolog predicates/programs; and
- design and implement correct, well-documented Prolog programs, utilising the language's execution model and programming principles.

The basic principles taught are: syntax and semantics, variables and atoms, unification and matching, resolution and execution, recursion, list processing, higher order predicates, and graph search.

Each lab session has a number of exercises that the students need to solve. They attempt them either at the lab or at home and, when they present a partial or complete solution to the instructor, they are given the link to the solutions in the form of a screencast. Thus, they have the opportunity to compare their solution to the model answer and complete it, based on what they see in the video.

Being familiar with learning technologies and taking into account various findings with regard to the effectiveness of screencasts in student learning [16], [19], we decided to use screencasts in which we solve specific exercises for students by showing them the Prolog way of thinking towards approaching a problem. Although PowerPoint slides can support animations, they are not particularly helpful for this purpose. The fact that a logic programme is not developed sequentially is rather difficult to be presented with PowerPoint animations and be accompanied with narration. Students enrolled in our Logic Programming course, offered in the second year of studies, already know Java and they face difficulties getting acquainted with the new programming paradigm. Our pool of screencasts is continuously enhanced with new material, exercises and solutions. Paying some attention not to contextualise the screencast with a particular group of students and avoiding mentioning dates or events, the screencasts can stay as a source of reference in the depository for years.

With screencasts we managed to show the whole process of developing a Prolog programme from scratch and at the same time "speak out loud" the way that someone should

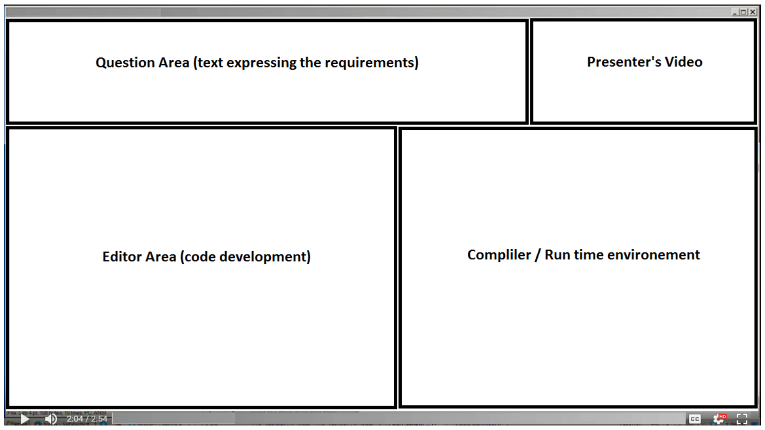

Question Area (text expressing the requirements)
Presenter's Video
Editor Area (code development)
Compliler / Run time environement
2:04 / 2:54

Fig. 4. A suggested screen layout for presenting the solution to a programming exercise

think in order to develop it. We took a step-by-step approach towards developing the programme, explaining how one should think, what the common pitfalls are, and also how to run the programme in order to visualize the results.

The screen was typically partitioned into three or four windows, as in Fig. 4:

- the exercise text which describes the programming task in question;
- the editor in which we show how the programme is developed;
- the execution environment; and
- the video of the presenter, visible at the beginning of the video and at any other point considered necessary.

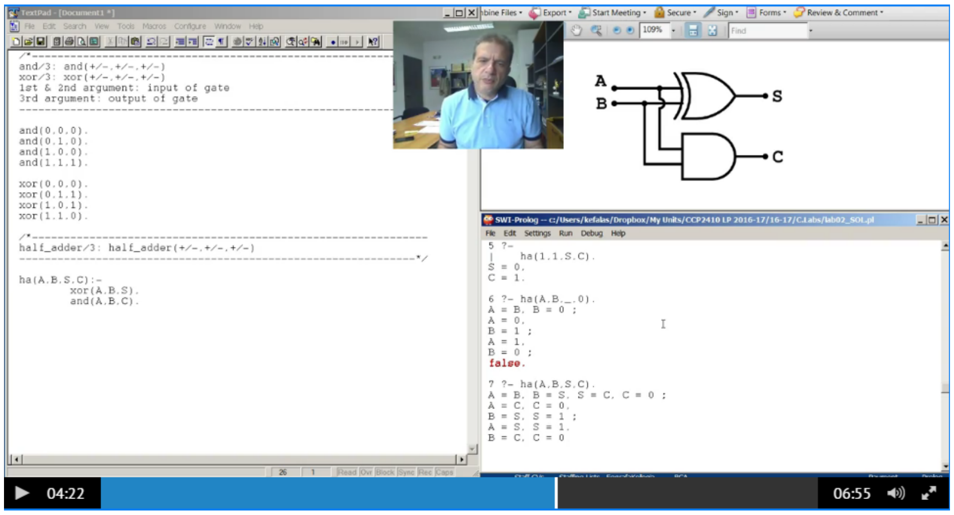
Figures 5 and 6 show two examples of screencasts for the development of a Prolog program with and without the presenter's video, respectively. We tried to follow all good practices listed in Section 5, most of them successfully. During the course of the past three years we have developed around 30–40 screencasts for our course, keeping their duration to no more than 5–6 minutes. Naturally, the more experience we gained, the less editing was required. We estimate that currently for a 5-minute screencast, we spend on average around 15 minutes overall.

As recording tools, we initially used the desktop version of echo360[6] and then Screencast-O-Matic. The distribution was made by uploading the link of the Screencast-O-Matic storage to our VLE.

8. Results

As a first step, we gathered statistics from our VLE, which show that the majority of the students watch the screencast exercise solutions. Actually, this is most likely an underestimation as the VLE's statistics cannot record the cases when students batch download the course's material and access it off-line.

[6] echo360.com



```
/*-----------------------------------------------------------
and/3: and(+/-,+/-,+/-)
xor/3: xor(+/-,+/-,+/-)
1st & 2nd argument: input of gate
3rd argument: output of gate
-------------------------------------------------------------

and(0,0,0).
and(0,1,0).
and(1,0,0).
and(1,1,1).

xor(0,0,0).
xor(0,1,1).
xor(1,0,1).
xor(1,1,0).

/*-----------------------------------------------------------
half_adder/3: half_adder(+/-,+/-,+/-)
-----------------------------------------------------------*/

ha(A,B,S,C):-
        xor(A,B,S),
        and(A,B,C).
```

```
5 ?-
|    ha(1,1,S,C).
S = 0,
C = 1.

6 ?- ha(A,B,_,0).
A = B, B = 0 ;
A = 0,
B = 1 ;
A = 1,
B = 0 ;
false.

7 ?- ha(A,B,S,C).
A = B, B = S, S = C, C = 0 ;
A = C, C = 0,
B = S, S = 1 ;
A = S, S = 1,
B = C, C = 0
```

TextPad - [Document1 *]
File Edit Search View Tools Macros Configure Window Help
SWI-Prolog -- c:/Users/kefalas/Dropbox/My Units/CCP2410 LP 2016-17/16-17/C.Labs/lab02_SOL.pl
File Edit Settings Run Debug Help
A
B
S
C
04:22
06:55

Fig. 5. Screencast screenshot including the presenter's video

Through the provided statistics, we have identified which screencasts have been the most popular. The top four were the screencasts referring to:

– list processing through recursion;
– revision exercises preparing students for the final exam;
– simple recursion methodology;
– basic Prolog syntax.

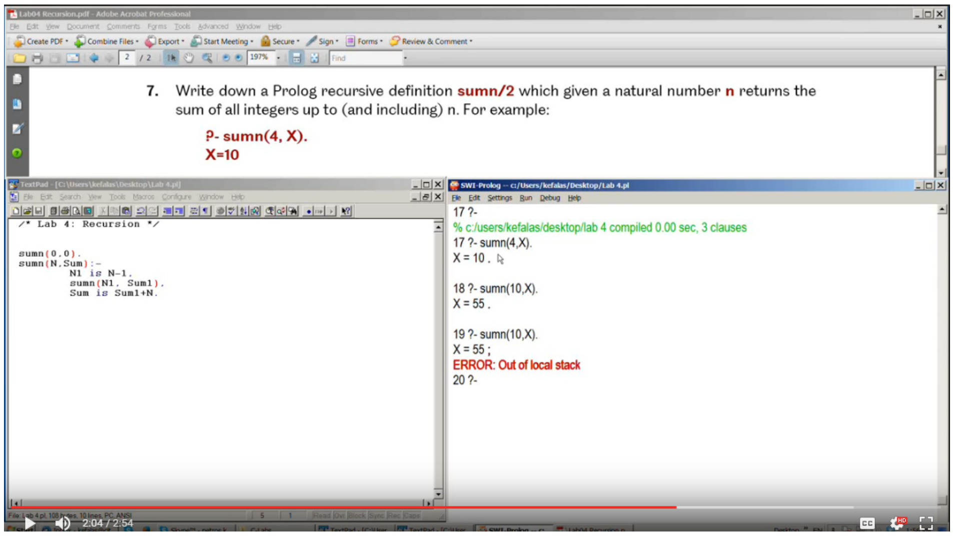
During private communications with students, they have many times informally expressed their satisfaction about the use of screencasts and have even, more formally, requested through their representatives during the Student-Staff Committee, that more screencasts are used in the context of other courses too. This was the incentive that drove us to the definition of our research question and the resulting questionnaire that was distributed to students who attended the course during the past three years.

In the first question a total of 96% of the students agree (summing the number of students who responded 'Strongly agree' and 'Agree') that screencasts have had a positive impact on their understanding of Logic Programming, with 62% feeling strongly about this (Fig. 7).

In the second question, although 91% of the students felt that the screencasts had a positive impact in their motivation to learn and practice (again summing the number of students who responded 'Strongly agree' and 'Agree'), only 57% felt strongly about it (Fig. 8).

The comments justifying the answers about students' understanding and motivation through screencasts are summarised below:

– "... easy, more direct, better way to understand the material taught";
– "... helpful to clarify things I did not understand in the class/lab";
– "... I can watch them at my own time and I can access them from anywhere";


Lab04 Recursion.pdf - Adobe Acrobat Professional
TextPad - [C:\Users\kefalas\Desktop\Lab 4.pl]
SWI-Prolog -- c:/Users/kefalas/Desktop/Lab 4.pl
2:04 / 2:54

7. Write down a Prolog recursive definition **sumn/2** which given a natural number **n** returns the sum of all integers up to (and including) n. For example:

```
?- sumn(4, X).
X=10
```

```
/* Lab 4: Recursion */


sumn(0,0).
sumn(N,Sum):-
        N1 is N-1,
        sumn(N1, Sum1),
        Sum is Sum1+N.
```

```
17 ?-
% c:/users/kefalas/desktop/lab 4 compiled 0.00 sec, 3 clauses
17 ?- sumn(4,X).
X = 10 .

18 ?- sumn(10,X).
X = 55 .

19 ?- sumn(10,X).
X = 55 ;
ERROR: Out of local stack
20 ?-
```

Fig. 6. Screencast screenshot without the presenter's video

– "... helpful to revise and compare my solutions to those given by the professors";
– "... taught me how to solve similar problems";
– "... videos are easier to comprehend than text";
– "... accessible at anytime needed!";
– "... helped me understand/taught me how to solve similar problems";
– "... helped me to see the lecturer working on the examples like in the classroom";
– "... was not very exciting but I understood its worth";
– "... helped me a lot with my revision";
– "... it was like the lecturer was with me while I was solving the exercises, and my questions were answered through the screencasts";
– "... screencasts are in a way an enhanced version of a whiteboard";
– "... I personally prefer writing my own notes so that I keep the structure that I want while revising. The screencasts, though, do not affect my notes in a negative way and allow me to study any missed lectures";
– "... it was not just pure code but training of thought".
– "... precise, concise, and simple";
– "... the explanations given while approaching the solution";
– "... the simplicity of the solutions after having watched the screencast";
– "... very interactive";
– "... the way that material was summarised";
– "... saved me a lot of time from keeping notes";
– "... I could watch them as many times as I wanted to";
– "... showing directly the code and implementation while explaining";
– "... sometimes when I do not manage to keep notes effectively screencasts saved me lots of time when I was studying";
– "... the problems required a bit more attention which made them more interesting".

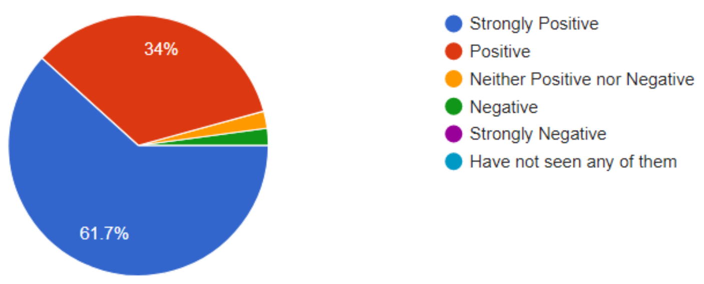


Fig. 7. Percentage of students who believe that screencasts impacted their understanding of Logic Programming

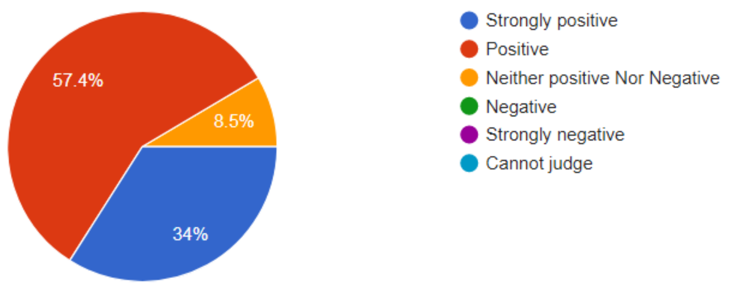


Fig. 8. Percentage of students who believe that screencasts impacted their motivation to learn

There is nothing much the students would change. The majority said “nothing” while many of them suggested “more and harder exercises” and few “some improvement on the quality of sound”.

100% of the participants agreed that “screencasts would be useful for other courses too”. The majority of them referred to technical courses where problem solving is required, especially programming. Few, but not an insignificant number, said that “screencasts could be helpful in all courses”.

The course in terms of intended learning outcomes and material has remained more or less the same with minor variations over the last decade. As aforementioned, it is mainly a skills course that aims to enhance the students’ way of thinking about programming. Screencasts have been available during the past three years, however their number was small during the first year and was later increased to cover all topics in the last two years.

Table 1 shows the results of students in the final examination, which is an assessed lab assessing all taught material, as well as the success (pass) rate. Cohorts from 2015 onwards were provided with screencasts, whereas the previous 2014–15 was not. We are not in a position, unfortunately, to compare with performance of earlier cohorts as the assessment scheme was significantly different before 2014. It should also be noted that students of the first of the three cohorts (2015–16):

- completed the questionnaire one year after they attended the course, concurrently with the 2016–17 cohort, and
- students of the first cohort had access to a limited pool of screencasts covering only a small number of topics in relation to all the content and learning outcomes. Our screencasts were substantially increased during the last two years.

As can be seen, students of the 2015–16 cohort demonstrated an overall lower performance but this was a more general issue apparent in all courses. As such, it is quite difficult to conclude anything about the impact of the small number of screencasts that was available to them. On the contrary, the last two cohorts, with the substantially increased number of screencasts, show increased performance, either in the exam average or in the success rate. Considering the three cohorts from 2015 onwards, the average in the final exam increases from 42% to almost 60%. Note that in the UK grading scheme, 40% is the bare pass while 70% signifies excellent performance. Likewise, the success rate has increased from 50% to 79%, which is quite impressive for the particular course.

Table 1. Performance of students during the last four years. Screencasts are available during the last three years

Academic Year	Performance in final Exam (average)	Success (Pass) Rate
2017-18	58.8	79%
2016-17	50.8	73%
2015-16	41.8	50%
2014-15	55.8	63%

9. Staff Development

All research shows that the use of technology in general, and screencasts in particular, can greatly enhance the student learning experience, further support weaker students, engage and motivate a larger part of the student body of a class. Academic staff in universities generally concede that a number of screencasts, for any of the reasons listed in this paper, would be an asset for any course.

However, the majority of colleagues have not employed screencast technology in their courses. This may be attributed to the following:

- they are not aware of the technology;
- they are aware of the technology and its benefits, but do not possess the technical skills;
- they perceive the value as unworthy of the effort;
- they believe that their course is not suitable;
- they are not willing to spend time on training and development.

The above reasons, of course, do not apply only to screencasts but overall, to a wider resistance in adopting any new learning technology.

Naturally, using a new technology requires ample time on the side of the lecturer, primarily in terms of getting familiar and confident with it, i.e getting past the initial learning curve. Many institutions, including ours, lately consider TEL as an important strategic goal and the best way to promote it is by providing the necessary support to staff who may be willing to experiment with new technologies. This may be in the form of staff development seminars or through some form of best practices dissemination. The importance of this support is also evident considering that the majority of people is inherently resistant to change.

We have come to understand that our colleagues do not need to be convinced on the effectiveness of screencasts, but they find it hard to get started. Practical hands-on experience is required before they feel confident about the result. They prefer personal tutoring rather than a general seminar, and we have succeeded in providing individual support to anyone who wishes to use this learning technology.

10. Conclusions

In this paper we have presented screencasts as a learning technology that can facilitate and enhance learning. We have provided some tips for creating screencasts for educational purposes, discussed the rationale behind their use, and shared our experience with using screencasts in programming. More particularly, we use screencasts as a technology enhanced learning method in a logic programming course. The latter may be considered an ideal case for using screencasts, due to its nature, but their use can certainly be affective in a variety of other courses.

Results from a questionnaire that was distributed to and completed by our students clearly demonstrate that screencasts have been very well-received, with the majority of students perceiving that screencasts have increased both their understanding of the programming paradigm and their motivation towards learning it. It was actually quite rewarding to see that students are asking for the use of screencasts in other courses as well. In addition, the performance of the students and, in consequence, the success rate of the course appear to be increasing, demonstrating that the approach has a positive impact in student learning. Our experience with using screencasts has overall been very rewarding; we plan to expand our existing repository for the logic programming course and, indeed, start using them in other courses too.

Future work should include learner analytics, such as the statistical analysis of the time intervals that students paid more attention to within a video. This can help us reach conclusions for the quality of the material presented, the way it is presented, and the level of comprehension of the students. As a consequence, our teaching can focus on the issues identified and, if necessary, we can edit existing screencasts or create additional ones. Finally, the inclusion of in-video questions is a challenge that, if implemented, could facilitate self-assessment and consequently enhance student learning.

References

1. ACM/IEEE-CS Joint Task Force on Computing Curricula: Computer Science Curricula 2013. Tech. rep., ACM Press and IEEE Computer Society Press (December 2013), `http://dx.doi.org/10.1145/2534860`
2. Brame, C.J.: Effective educational videos: Principles and guidelines for maximizing student learning from video content. CBELife Sciences Education 15(4), es6 (2016)
3. Chen, C.M., Wu, C.H.: Effects of different video lecture types on sustained attention, emotion, cognitive load, and learning performance. Computers & Education 80, 108–121 (2015)
4. Cummins, S., Beresford, A.R., Rice, A.: Investigating engagement with in-video quiz questions in a programming course. IEEE Transactions on Learning Technologies 9(1), 57–66 (2016)
5. Donkor, F.: Assessment of learner acceptance and satisfaction with video-based instructional materials for teaching practical skills at a distance. The International Review of Research in Open and Distributed Learning 12(5), 74–92 (2011)
6. Ghilay, Y.: Math courses in higher education: Improving learning by screencast technology. GSTF Journal on Education (JEd) 4(2) (2018)
7. Ghilay, Y., Ghilay, R.: Computer courses in higher-education: Improving learning by screencast technology. Journal of Educational Technology 11(4), 15–26 (2015)
8. Giannakos, M.N., Chorianopoulos, K., Chrisochoides, N.: Making sense of video analytics: Lessons learned from clickstream interactions, attitudes, and learning outcome in a video-assisted course. The International Review of Research in Open and Distributed Learning 16(1) (2015)
9. Green, K.R., Pinder-Grover, T., Millunchick, J.M.: Impact of screencast technology: Connecting the perception of usefulness and the reality of performance. Journal of Engineering Education 101(4), 717 (2012)
10. Guo, P.J., Kim, J., Rubin, R.: How video production affects student engagement: an empirical study of mooc videos. In: Proceedings of the first ACM conference on Learning@ scale conference. pp. 41–50. ACM (2014)
11. Guo, P.J., Kim, J., Rubin, R.: How video production affects student engagement: An empirical study of mooc videos. In: Proceedings of the first ACM conference on Learning@ scale conference. pp. 41–50. ACM (2014)
12. Hansch, A., Hillers, L., McConachie, K., Newman, C., Schildhauer, T., Schmidt, P.: Video and online learning: Critical reflections and findings from the field. SSRN eLibrary (2015), `http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2577882`
13. Kefalas, P., Stamatopoulou, I.: Using screencasts to enhance logic programming skills. In: Proceedings of the 8th Balkan Conference in Informatics (BCI2017), 20–23 September 2017, Skopje. ACM, New York, NY, USA (2017)
14. Ketikidis, P.H., Ververidis, Y., Kefalas, P.: The case of the University of Sheffield (TUOS) International Faculty, CITY College: An example of an entrepreneurial model for internationalisation of higher education. UIIN Good Practice Series: Fostering University-Industry Relationships, Entrepreneurial Universities and Collaborative Innovation pp. 53–70 (2013)
15. Lee, M.J., Pradhan, S., Dalgarno, B.: The effectiveness of screencasts and cognitive tools as scaffolding for novice object-oriented programmers. Journal of Information Technology Education: Research 7, 61–80 (2008)
16. Mohorovičić, S., Tijan, E.: Using screencasts in computer programming courses. In: Proceedings of the 22nd EAEEIE Annual Conference, Maribor. pp. 220–225 (2011)
17. Morris, C., Chikwa, G.: Screencasts: How effective are they and how do students engage with them? Active Learning in Higher Education 15(1), 25–37 (2014)
18. O'Flaherty, J., Phillips, C.: The use of flipped classrooms in higher education: A scoping review. The internet and higher education 25, 85–95 (2015)

19. Pal, Y., Iyer, S.: Effect of medium of instruction on programming ability acquired through screencast. In: Learning and Teaching in Computing and Engineering (LaTiCE), 2015 International Conference on. pp. 17–21. IEEE (2015)
20. Powell, L.M., et al.: Evaluating the effectiveness of self-created student screencasts as a tool to increase student learning outcomes in a hands-on computer programming course. Information Systems Education Journal 13(5), 106 (2015)
21. Rackaway, C.: Video killed the textbook star?: Use of multimedia supplements to enhance student learning. Journal of Political Science Education 8(2), 189–200 (2012)
22. Reinecke, D., Finn, L.: Video lectures in online graduate education: Relationship between use of lectures and outcome measures. Journal of Information Technology Education: Research 14(1), 113–121 (2015)
23. San Miguel, B., Aguirre, S., del Álamo, J.M., Cortés, M.: A proposal for enhancing the motivation in students of computer programming. ICERI2012 Proceedings pp. 1157–1164 (2012)
24. Schwan, S., Riempp, R.: The cognitive benefits of interactive videos: learning to tie nautical knots. Learning and Instruction 14(3), 293–305 (2004)
25. Secker, J., Bond, S., Grussendorf, S.: Lecture capture: rich and strange, or a dark art? (2010), `http://eprints.lse.ac.uk/29184/`
26. Sharp, J.H., Schultz, L.A.: An exploratory study of the use of video as an instructional tool in an introductory c# programming course. Information Systems Education Journal 11(6), 33 (2013)
27. Sharp, J.H., Schultz, L.A.: An exploratory study of the use of video as an instructional tool in an introductory c# programming course. Information Systems Education Journal 11(6), 33 (2013)
28. Smith, J.G., Smith, R.L.: Screen-capture instructional technology: A cognitive tool for designing a blended multimedia curriculum. Journal of Educational Computing Research 46(3), 207–228 (2012)
29. Stigler, J., Geller, E., Givvin, K.: Zaption: A platform to support teaching, and learning about teaching, with video. Journal of E-Learning and Knowledge Society 11(2) (2015)
30. Sun, L., Frederick, C., Sanjuan Espejo, P., Cunningham, R.: Can we teach a programming language as a second language? Computer in Education 7(3) (2016)
31. Tiernan, P.: An inquiry into the current and future uses of digital video in university teaching. Education and Information Technologies 20(1), 75–90 (2015)
32. Triantafyllou, E., Timcenko, O.: Student perceptions on learning with online resources in a flipped mathematics classroom. In: CERME 9-Ninth congress of the European society for research in mathematics education. pp. 2573–2579 (2015)
33. Turmel, W.: 10 Steps to Successful Virtual Presentations. 10 steps to success, ASTD Press (2011), `https://books.google.gr/books?id=qezqmy8tcOMC`
34. Walker, J., Cotner, S., Beermann, N.: Vodcasts and captures: Using multimedia to improve student learning in introductory biology. Journal of Educational Multimedia and Hypermedia 20(1), 97–111 (2011)
35. Yoon, C., Oates, G., Sneddon, J.: Undergraduate mathematics students reasons for attending live lectures when recordings are available. International Journal of Mathematical Education in Science and Technology 45(2), 227–240 (2014)

Professor Petros Kefalas is the University of Sheffield International Faculty Director of Learning and Teaching. He holds an MSc in Artificial Intelligence and a PhD in Computer Science, both with the University of Essex. He has conducted research in Parallel Logic Programming, Artificial Intelligence, Formal Methods and Multi-Agent Systems, co-authored a Greek textbook in Artificial Intelligence, and is currently involved in investigating the applicability of formal methods for modelling, verifying, and testing multi-agent systems with emergent and emotional behaviour. He is a member of BCS, ACM, IEEE, ALP, the Greek Computer Society (EPY), and the Hellenic AI Society.

Ioanna Stamatopoulou is a Lecturer at the Computer Science Department of the University of Sheffield International Faculty. She holds an MSc in Artificial Intelligence with the University of Edinburgh and a PhD in Computer Science with the University of Sheffield. She is a member of the Intelligence, Modelling and Computation (IMC) Research Group of the department and a member of the Hellenic Artificial Intelligence Society. Her research interests currently lie in the area of modelling and simulating emotional agents and multi-agent systems.

Received: March 12, 2018; Accepted: September 3, 2018.