



Article

Crossing the Borders: Re-Use of Smart Learning Objects in Advanced Content Access Systems

Hamza Manzoor ¹, Kamil Akhuseyinoglu ², Jackson Wonderly ¹, Peter Brusilovsky ² and Clifford A. Shaffer ^{1,*}

¹ Department of Computer Science, Virginia Tech, Blacksburg, VA 24061, USA

² School of Computing and Information, University of Pittsburgh, 135 North Bellefield Avenue, Pittsburgh, PA 15260, USA

* Correspondence: shaffer@vt.edu

Received: 31 May 2019; Accepted: 15 July 2019; Published: 19 July 2019



Abstract: Researchers in many disciplines are developing novel interactive smart learning objects like exercises and visualizations. Meanwhile, Learning Management Systems (LMS) and eTextbook systems are also becoming more sophisticated in their ability to use standard protocols to make use of third party smart learning objects. But at this time, educational tool developers do not always make best use of the interoperability standards and need exemplars to guide and motivate their development efforts. In this paper we present a case study where the two large educational ecosystems use the Learning Tools Interoperability (LTI) standard to allow cross-sharing of their educational materials. At the end of our development process, Virginia Tech’s OpenDSA eTextbook system became able to import materials from Aalto University’s ACOS smart learning content server, such as python programming exercises and Parsons problems. Meanwhile, University of Pittsburgh’s Mastery Grids (which already uses the ACOS exercises) was made to support CodeWorkout programming exercises (a system already used within OpenDSA). Thus, four major projects in CS Education became inter-operable.

Keywords: computer science education; learning tools interoperability; etextbook; smart learning objects

1. Introduction

Computer Science Education Researchers have a long history of creating online educational materials. As might be expected, systems that support practicing small programming exercises have been available for many years, for example, Coding Bat [1] and CodeWorkout [2]. But many other problem types have been developed and deployed successfully, such as for Parson’s Problems [3] and data structures proficiency exercises [4]. Likewise, visualizations have been used for decades to show the dynamic behavior of algorithms [5–7].

There exist communications protocols that allow third-party educational tools to integrate with Learning Management Systems (LMS). Learning Tools Interoperability (LTI) [8] is one such specification. It was developed by IMS Global Learning Consortium and is now adopted by all major LMS. The purpose of LTI is to establish a standard way to securely integrate remotely hosted learning applications with platforms like an LMS.

Under LTI terminology, learning applications are called Tools and they are delivered by Tool Providers. A tool might be an interactive exercise that a student might do for a grade. An LMS (systems like Moodle, Canvas or Blackboard) is called a *Tool Consumer*. The LTI specification enhances the native functionality provided by a Tool Consumer to include the tools made available by the tool provider. The result is that instructors have available many “smart objects” that they can embed into

their course. Functionally, with LTI the LMS can display the third-party content to a student as though it were a native part of the LMS, collect scoring results from the third-party content and store this result in the LMS gradebook once the student has completed it.

Many learning applications now support LTI but usually this integration is simple and uni-directional, with an LMS consuming small “smart learning objects” like a small exercise or visualization. But education researchers are developing much richer content and tools that require more sophisticated communications structures than this simple uni-directional model allows. In particular, eTextbook systems such as OpenDSA [9,10] or Runestone [11] represent learning content as a hierarchically structured textbook and marshal a range of content types into an integrated whole. Another example is personalized content access portals such as SQL-Tutor [12] or MasteryGrids [13], which attempt to guide students to the most appropriate interactive activities by modeling the student’s knowledge and visualizing their learning progress. Such systems should not try to duplicate the administrative services provided by an LMS, such as authentication of users and maintaining scores but at the same time they need to make decisions based on previous history of the students’ interactions. So the simplest model of a smart learning object communicating with an LMS breaks down in these situations. Further, both the “smart objects” and the eTextbook itself typically generates a rich set of learner analytics data. This information is easily lost if it can only be communicated to the LMS. Better is to think of a modern educational system as an ecosystem of collaborating components, with communications interdependancies that cannot be known until they are brought together.

In this paper we are trying to move one step closer to this complex ecosystem by presenting a case of more advanced integration, which allows multiple kinds of interactive learning content developed by four different teams to be used transparently with two different advanced content access platforms—OpenDSA and Mastery Grids. In Section 2 we describe in some detail the integration goals of the case study. In Section 3 we detail the four major components involved in the case study: OpenDSA, Mastery Grids, CodeWorkout and the ACOS Advanced Content Server. Section 4 presents technical details for how the integration is implemented. Section 5 presents our reflections on the experience.

2. The Integration Goal

The OpenDSA eTextbook system [9] is an example of a growing number of efforts to provide a variety of online educational materials. OpenDSA supports a range of Computer Science courses with content on a growing list of topics, including Data Structures and Algorithms, Formal Languages and Translators and Compilers. OpenDSA allows users to choose from available modules to put together a custom textbook for a course. The custom book can be exported to Canvas as individual modules and assignments. OpenDSA is an open-source project with visualizations and exercises authored by many collaborators and volunteers. OpenDSA also makes use of small programming exercises provided by a separate system, CodeWorkout [2]. By adopting exercises from CodeWorkout, the developers of OpenDSA are not responsible for creating and maintaining a technically demanding exercise type but are still able to make these exercises available to students.

Mastery Grids [13] is an open-source personalized content access interface designed at the University of Pittsburgh. It has been used with a variety of courses including Python programming, Java programming and Databases. The system allows instructors to design a course-adapted interface to several types of interactive learning content hosted on independent content servers. Among these servers, ACOS Server [14] provides access to Java and Python program animations as well as Parson’s exercises aimed at introductory programming classes.

It so happens that OpenDSA (and CodeWorkout) on the one hand and Mastery Grids with ACOS on the other hand, provide fairly disjoint collections of materials. OpenDSA focuses on advanced programming and visualizations of materials typically used at the sophomore level and above, while the materials developed for Mastery Grids tends to support the topics offered in first semester programming courses. But these divisions are not hard, nor permanent, in that some materials

used in one system could be of benefit to classes whose topical content might otherwise favor use of the other system. Likewise, there are differences in the domains covered by the systems. For example, content available through Mastery Grids supports Python-based courses, while OpenDSA currently focuses on content for courses in Java and C++.

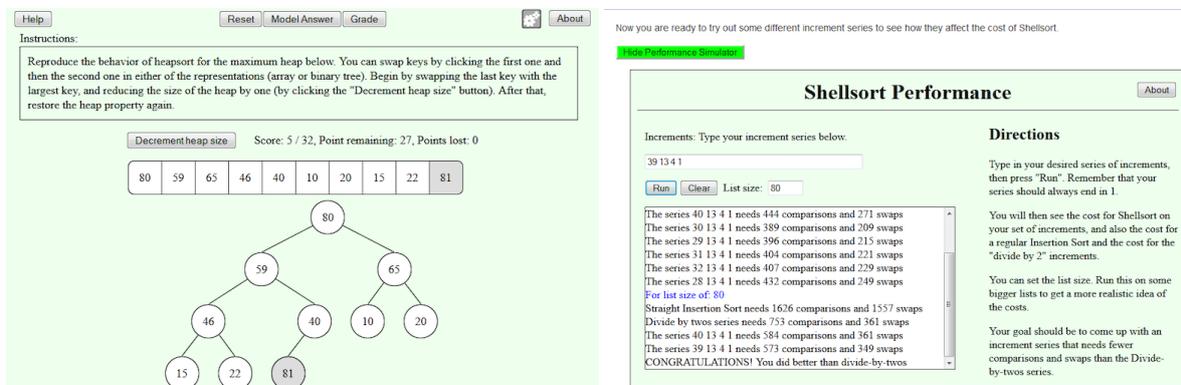
One solution to expanding the resources available in either system is to re-implement existing materials. But this would be an inefficient approach. Even if the materials were directly ported from their original source (which would reduce development cost over reinventing entirely from scratch), maintainability would be poor. Better is to integrate existing content from the various systems, allowing the original developers of a given exercise or content type to keep improving to the benefit of all. Instead, we see to support cross-integration of materials in the two infrastructures. The key idea explored by our teams is leveraging the LTI protocol to design an inter-operable connection between various kinds of advanced content across systems and smart learning content. While LTI is not typically used to support this kind of integration, our work demonstrates that it can successfully support our more complex scenario. To demonstrate the feasibility of the new approach, we made use of the Tool-Consumer capabilities of OpenDSA to integrate animations and exercises from ACOS. We also extended the contents served by Mastery Grids to incorporate programming exercises available at CodeWorkout. The next section explains each of these tools in order to better understand the integration that was done, the process used to achieve it and its outcomes.

3. The Integration Components

3.1. OpenDSA

OpenDSA is an open-source eTextbook project lead by Virginia Tech, with many collaborators from around the world. OpenDSA materials include hundreds of visualizations and interactive exercises that support courses in a wide variety of Computer Science-related topics such as Data Structures and Algorithms, Formal Languages and Programming Languages [9,10]. Interactive algorithm visualizations illustrate most algorithms and data structures in the OpenDSA collection. OpenDSA combines textbook-quality prose with interactive visualizations and exercises and lets students practice as much as they need. It also allows students to control the pace of the visualization and to enter their own test cases to see how the algorithm or data structure works on their input. OpenDSA exercises provide immediate feedback to students at every step.

Figure 1a shows an example of an exercise in OpenDSA. Here, students reproduce the behavior of the algorithm to create a maximum heap by swapping records in an array. Similarly, Figure 1b shows a simulation of the performance of shellsort. Students can analyze the cost of shellsort on their own custom series of diminishing increments.



(a) Exercise to reproduce the behavior of a maxheap (b) Simulator to analyze the performance of shellsort

Figure 1. OpenDSA visualizations and exercises.

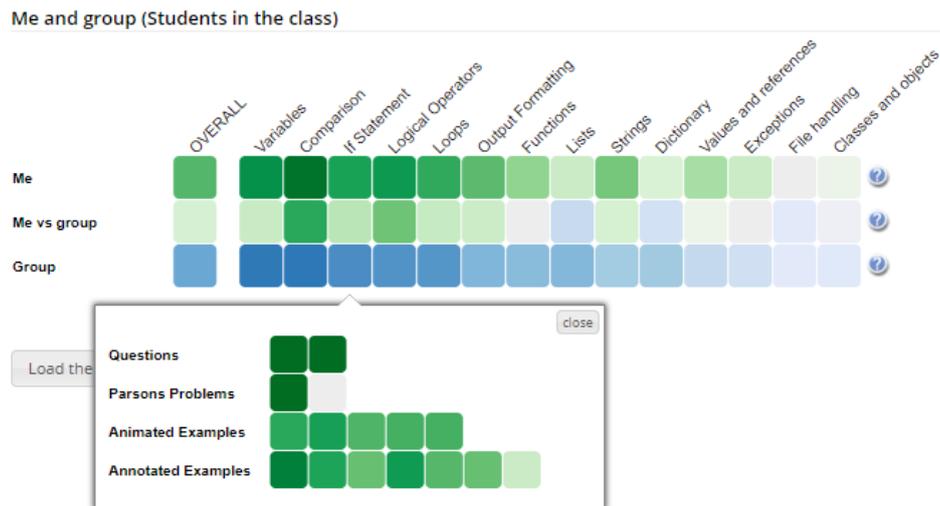


Figure 3. Mastery Grids interface showing student progress at the topic level and the available smart learning contents for the *If Statement* topic. Student’s progress for each content type is shown in the pop-up window.

3.3. CodeWorkout

CodeWorkout [2] is an online drill-and-practice system developed at Virginia Tech, primarily to provide students with small-scale programming assignments. It allows students to develop experience and knowledge by completing code-writing activities and answering multiple-choice questions. CodeWorkout provides some course management functionality, like the ability to manage assignments (deadlines, grades, weights, etc.) and users (enrollment, authentication, extensions, etc.). However, it is possible that an instructor would want to use CodeWorkout in their course along with other online educational materials, which likely provide similar course management features. Having to manage these tasks on multiple fronts (e.g., Canvas, OpenDSA and CodeWorkout) can be a daunting endeavour, possibly resulting in reduced rates of adoption.

To facilitate CodeWorkout’s use within a more general context, its developers have made CodeWorkout into an LTI-compliant *tool provider*. This means that CodeWorkout exercises can be practiced on the CodeWorkout site itself or they can be embedded inside other web applications (i.e., *tool consumers* like Canvas). LTI allows the tool consumer (such as OpenDSA or an LMS) to authenticate users, serve them CodeWorkout exercises and receive and record their attempts and grades, all without requiring the user to manually sign in to or open CodeWorkout and requiring minimal management effort from course staff. In later sections, we describe how we integrated CodeWorkout exercises into two LTI-compliant tool consumers: OpenDSA (Figure 8) and Mastery Grids (Figure 12).

3.4. ACOS: Advanced Content Server

ACOS server is a smart learning content server developed as a joint project between Aalto University and the University of Pittsburgh [14]. It enhances the re-usability of online learning activities by decoupling the content types from the interoperability protocols. This enables content developers to concentrate on the content creation without dealing with the details of the protocols.

ACOS server currently supports multiple communication protocols including LTI, A+ protocol [16] (a lightweight protocol similar to LTI), and the ADAPT2 protocol [17] (a similar protocol that focuses on adaptive learning content). ACOS also supports the standard HTML protocol, allowing learning activities to be embedded into a webpage (but this action does not communicate scoring or analytics to any consumer).

ACOS hosts multiple types of learning content that are valuable for beginner-level programmers and covers a full set of introductory programming topics ranging from variables to recursion and classes. In our initial integration, we concentrated on Parson’s problems [3] and code execution visualizations. The code execution visualizations are animated examples that demonstrate how programming constructs are executed and how the program state is changed after each execution step (Figure 4). They are implemented by the Jsvee library [18]. Parson’s problems are a form of coding exercise where students construct a program by selecting from a collection of given code fragments. Parson’s problems (Figure 5) are implemented by using the Js-parsons library [19]. As mentioned before, both Parson’s problems and visualizations are already accessible from the Mastery Grids interface (Figure 6). In our current work, we made both types accessible from OpenDSA (Figures 9 and 10) using the LTI protocol. The details are explained in Section 4.2.

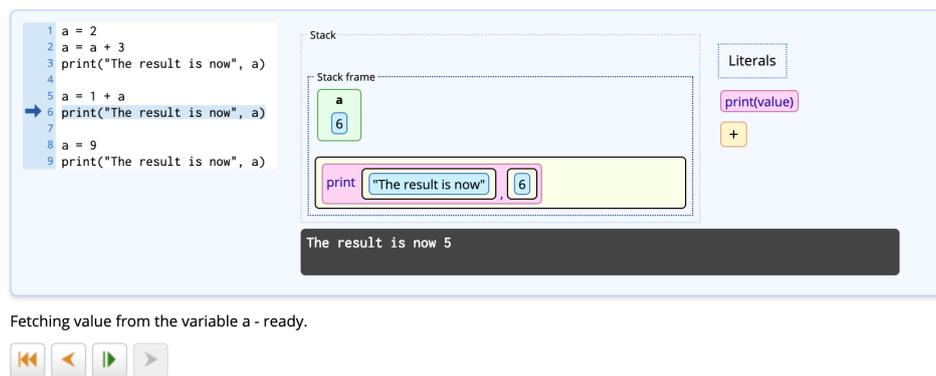


Figure 4. An example of program visualization (animated example) that is hosted by ACOS server. The visualization shows the stack state and the console output based on the current execution step.

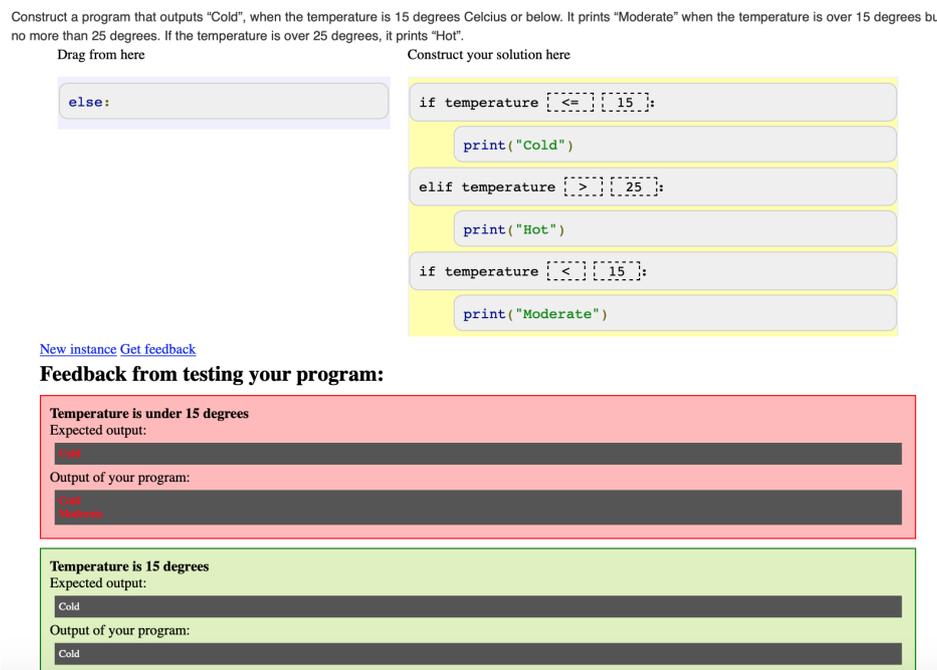


Figure 5. An instance of ACOS Parson’s problem, which asks students to construct a program to indicate weather conditions based on the temperature.

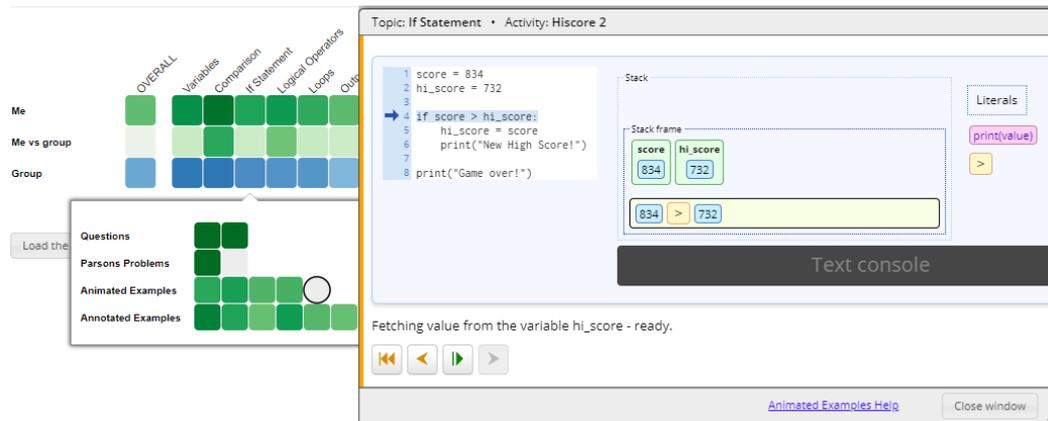


Figure 6. A program visualization (animated example) loaded from the Mastery Grids interface.

4. Materials and Methods

4.1. OpenDSA as a Tool-Consumer

Historically, OpenDSA started as only an LTI tool provider. OpenDSA modules could be added to Canvas through the Canvas API. Over time, instructors wanted to add support for coding exercises in OpenDSA modules. Our original design involved using CodeWorkout [2] as another tool provider in Canvas. To accomplish this, OpenDSA content creators would add CodeWorkout exercises as a part of the OpenDSA module. Then instructors would create their courses on OpenDSA and OpenDSA would populate the Canvas course through the Canvas API. The Canvas API calls would not only create OpenDSA modules in Canvas but also would create CodeWorkout assignments separately. Unfortunately, this goes against the model of “OpenDSA as a textbook” that has understanding and control of its exercises (while communicating necessary information to the LMS). Instead, separate Canvas assignments are required, which means that students had to navigate away from the OpenDSA module and open the assignment in a separate page. Also, OpenDSA did not have a complete picture of students’ performance, because under this model CodeWorkout and Canvas directly communicated with each other. Finally, OpenDSA could not group multiple assignments onto a page as might be the more natural presentation, because Canvas requires that each of its assessments be mapped to separate Canvas pages.

To overcome these problems, the OpenDSA development team implemented a new architecture that allows OpenDSA to act as both a tool provider and a tool consumer. This new architecture enables OpenDSA to integrate with other external tools. In this particular example, OpenDSA uses CodeWorkout as an external tool and includes the CodeWorkout assignments within its modules, as shown in Figure 7. Now, Canvas uses OpenDSA as an external tool and has no direct interaction with or information about, CodeWorkout. Canvas sends an LTI launch request to OpenDSA and then OpenDSA sends an LTI launch request to CodeWorkout to display coding exercises in the textbook. CodeWorkout responds to the LTI launch request from OpenDSA by displaying the requested exercise, which is shown within an iframe in the OpenDSA module, while the OpenDSA module is itself displayed within an iframe in the Canvas page.

Figure 8 shows a CodeWorkout exercise embedded within an OpenDSA module as it appears within Canvas. This allows students to solve programming exercises from within the same Canvas page where they read prose and see visualizations. This also allows OpenDSA to have a more complete picture of how a student is performing on other exercises, since it knows when the CodeWorkout exercise has been accessed, as well as what scores students get on CodeWorkout exercises. We believe that this approach offers a richer model for the future of online textbook construction. While inclusion of external interactive content has been used in several online textbook projects [11,20], it was done using proprietary integration approaches making it impossible to use interactive content across

multiple projects. Using standard interaction protocols opens a way to a broad re-use of smart content. The approach used by OpenDSA, whereby intermediaries stand between the LMS and the individual smart object types, points the way to richer ecosystems of collaborating components. Unfortunately there still remains the problem that CodeWorkout does not share details about student interactions within the programming exercise with OpenDSA (or any other third party accessing it). This is a communications deficiency that deserves attention in future work but becomes no longer a problem inherent in the communications architecture.

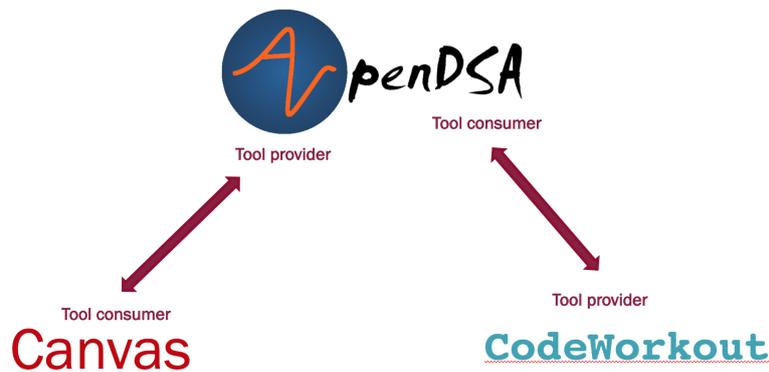


Figure 7. OpenDSA New Architecture.

operations that one naturally expects to perform on lists and serves to illustrate the issues relevant to implementing a structure. As an example of using the list ADT, here is a function to return true if there is an occurrence of a given integer and false otherwise. The find method needs no knowledge about the specific list implementation, just the list ADT.

```

// Return true if k is in list L, false otherwise
static boolean find(List L, Object k) {
    for (L.moveToStart(); !L.isAtEnd(); L.next())
        if (k == L.getValue()) return true; // Found k
    return false; // k not found
}
  
```

In languages that support it, this implementation for find could be rewritten as a generic or template with respect to the el. While making it more flexible, even generic types still are limited in their ability to handle different data types stored or particular, for the find function generic types would only work when the description for the object being searched for (k in t) is of the same type as the objects themselves. They also have to be comparable when using the == operator. A more realistic is that we are searching for a record that contains a key field whose value matches k. Similar functions to find and return a type based on a key value can be created using the list implementation, but to do so requires some agreement between and the find function on the concept of a key, and on how keys may be compared.

There are two standard approaches to implementing lists, the *array-based list*, and the *linked list*.

9.2.2. List ADT Programming Exercise

X278: ListADT

Use appropriate method calls from the List ADT to create the following list:
< 4 19 | 23 30 >

You should assume that L is passed to the function as an empty list.

```

public List buildList(List L)
{
}
  
```

Check my answer! Reset

Figure 8. A CodeWorkout exercise embedded within an OpenDSA module, displayed through Canvas.

4.2. Integrating ACOS Visualizations and Exercises into OpenDSA

While the integration of OpenDSA and CodeWorkout was done using a standard protocol, it was not a complete test of the new approach since both tools were developed in the same institution. To explore our approach across institutional borders, we moved to integrated ACOS server content into OpenDSA textbooks. The integration of OpenDSA and ACOS became possible because OpenDSA works as both a tool consumer and a tool provider, as explained above.

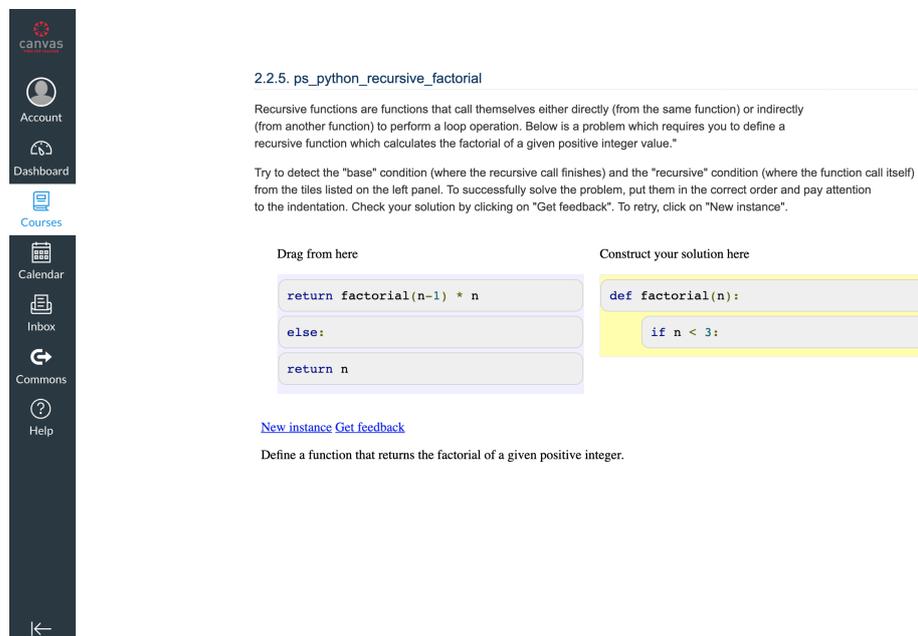
OpenDSA acts as a tool provider to Canvas to serve textbooks and within those books, OpenDSA calls other tools such as CodeWorkout to serve various coding exercises. We added ACOS as a new tool within OpenDSA to serve the Python exercises and animated examples for Java and Python. To do so, first we made minor modifications to the OpenDSA external tools script which allows instructors or content developers to add various exercises through external tools in their books. This script is called

during the book compilation process if external tools are used and it simply generates an HTML div element with data attributes that store information about the external tool exercise, including the ID assigned to the exercise by OpenDSA. We then added references to Parsons problems, Java animations and Python animations, respectively, into the OpenDSA database as LTI tool providers.

When an OpenDSA module that contains content from one or more of these tools is loaded in a user's browser, the OpenDSA front-end framework retrieves the ID of the exercise from a data attribute on the exercise's div element. The front-end framework then replaces the div element with an iframe whose target URL points to an endpoint on the OpenDSA server, with the URL containing the ID of the exercise as a parameter. When the iframe is rendered by the browser it results in a GET request being sent to the iframe's target URL. The OpenDSA server receives the request and retrieves the information for the tool associated with the exercise from the database, including the LTI launch URL of the tool. OpenDSA then generates the standard LTI launch request parameters, signs the request with OAuth using the key and secret that were stored with the tool information, then generates an HTML page containing a form with all of the LTI and OAuth parameters as fields. When this page is rendered by the user's browser, a single line of JavaScript automatically submits the form, which results in a POST request being sent to the tool's launch URL. The tool (e.g., ACOS) receives the request and delivers the exercise or animation which is displayed inside an iframe in the OpenDSA module.

The integration enabled 34 Parsons problems to be added to various OpenDSA books. These are graded exercises, so when OpenDSA sends a request to ACOS, it expects a score back from it once a student finishes the exercise. These scores are then passed on to Canvas and are shown in the Canvas gradebook. Figure 9 shows an example of a Parsons problem served through Canvas via the OpenDSA eTextbook.

Figure 10 shows an example of Python function animation served through Canvas via the OpenDSA eTextbook. Currently, 56 Python and 53 Java animations can be added to OpenDSA books.



2.2.5. ps_python_recursive_factorial

Recursive functions are functions that call themselves either directly (from the same function) or indirectly (from another function) to perform a loop operation. Below is a problem which requires you to define a recursive function which calculates the factorial of a given positive integer value.*

Try to detect the "base" condition (where the recursive call finishes) and the "recursive" condition (where the function call itself) from the tiles listed on the left panel. To successfully solve the problem, put them in the correct order and pay attention to the indentation. Check your solution by clicking on "Get feedback". To retry, click on "New instance".

Drag from here

```
return factorial(n-1) * n
else:
return n
```

Construct your solution here

```
def factorial(n):
    if n < 3:
```

[New instance](#) [Get feedback](#)

Define a function that returns the factorial of a given positive integer.

Figure 9. A Parsons problem (served by ACOS) made available within an OpenDSA eTextbook, delivered via Canvas.

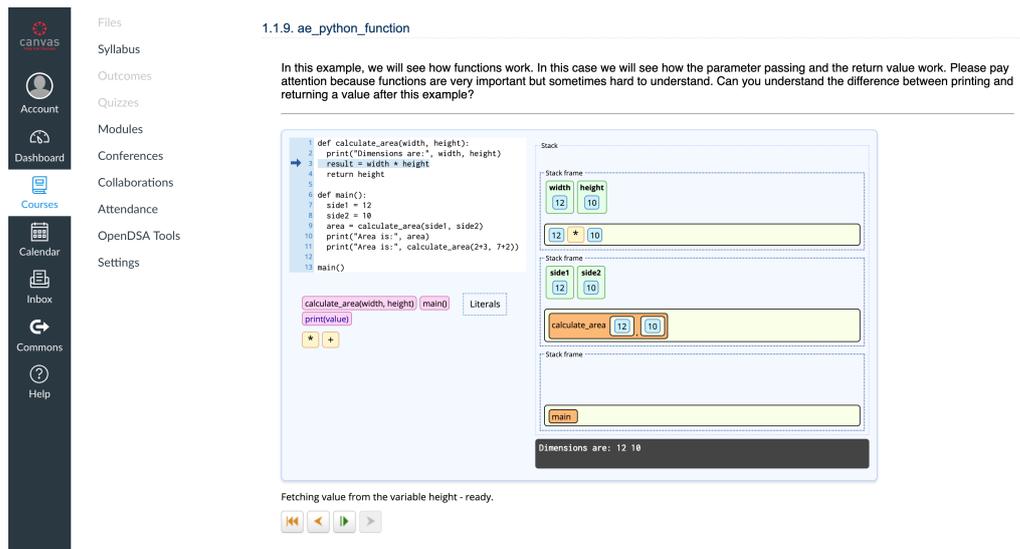


Figure 10. Python function animation (served by ACOS) made available within an OpenDSA eTextbook, delivered via Canvas.

4.3. Making Mastery Grids an LTI Consumer

To further confirm the potential of the new approach, we attempted to re-implement it in a different advanced content access tool, Mastery Grids [13]. The Mastery Grids interface originally served multiple types of learning content using the ADAPT2 proprietary content invocation protocol [17]. This loads a learning object (such as a coding exercise or Parson’s problem) into an iFrame by sending an HTTP GET request to the content provider. *Learning events* are generated based on learner actions at the content and reported back to a dedicated student model server through the ADAPT2 learner modeling protocol [17,21]. An important feature of the Mastery Grids interface is visualization of a learner’s topic-level knowledge progress. The necessary information is accessed by querying the student modeling server, which maintains a knowledge model of every student by processing the learning events sent by various types of content.

The integration process described above required development effort at the content provider side. Tool providers need to make sure that the generated learning events are successfully reported back to the student modeling server and additional development effort could be cumbersome for content providers with scarce development resources. The drawback of this integration process was already being addressed by the researchers at the University of Pittsburgh and Aalto University by supporting the proprietary ADAPT2 protocol in ACOS server as described in Section 3.4. However, this still requires content providers to make their contents accessible through ACOS server.

To implement the new communication approach, Mastery Grids was enhanced to support LTI Version 1.1, allowing it to function as an LTI consumer similar to an LMS, while keeping its distinctive features such as the progress visualization provided through the student modeling service.

We implemented our own LTI consumer service as a NodeJS server application, used to send LTI launch requests on behalf of the Mastery Grids interface. Implementing a separate LTI service decouples the LTI requirements from Mastery Grids and enables Mastery Grids to serve content through multiple protocols including LTI. This service is called when a content cell is clicked on the interface. Mastery Grids sends the unique user id and the requested content information to the LTI consumer service. Based on the requested content and the user information, the service determines which LTI tool provider will be called and generates the corresponding LTI launch request. Such an LTI launch request holds information to identify the requested content, the user and an LTI outcome service URL. This outcome service listens to the LTI grade callbacks generated by the LTI tool. The service authenticates itself by signing the launch request with the consumer secret provided by the LTI tool provider using OAuth 1 libraries available in JavaScript. Then, the launch request is

sent via HTTP POST request to the tool provider. When the tool provider receives the launch request, it checks the validity of the request and loads the content if the request is valid.

As described in the LTI standards, we also implemented an LTI outcome service that listens to the LTI grade callbacks generated from the LTI tool providers. The outcome service URL is called when a user performs a graded action while interacting with an LTI tool. The service URL should be publicly accessible and needs to be passed as a post-request parameter in the LTI launch request as described above. The grade callback message includes required information about the learning content, the user and the action result (i.e., if the answer was correct or incorrect). Then, the reported grade result is directed to the student modeling service using the ADAPT2 protocol. Thus, LTI compatibility is achieved by an *adapter pattern* approach, keeping the existing features of the Mastery Grids interface and the student modeling service.

4.4. Integrating CodeWorkout Exercises into Mastery Grids

LTI compatibility makes Mastery Grids a fully-fledged smart content delivery platform as defined in Reference [22]. Being an LTI consumer eliminates further development requirements on the provider side and opens the way to serve any LTI-compatible learning content. To demonstrate the opportunities provided by the new approach, we performed the integration of CodeWorkout exercises into Mastery Grids. We selected CodeWorkout as an exemplary tool provider because of its diverse set of programming exercises and existing LTI compatibility. This allowed Mastery Grids to provide programming practice from existing exercises at a variety of content levels. In contrast, developing a new coding tool and authoring coding exercises at the same quality would require a considerable amount of time and would be another bad example of re-implementation of an existing learning tool.

To show the feasibility of our approach, we added CodeWorkout exercises to an existing Java course in Mastery Grids and nearly doubled the number of practice problems available to the students (Figure 11). In this course, learners have access to multiple learning activities such as animated examples (as mentioned in Section 3.4), program construction examples and challenges [23], semantic problems [24] and newly integrated CodeWorkout coding exercises. Currently, the CodeWorkout exercises are linked to Mastery Grids manually. However, as future work, we plan to implement the *LTI Deep Linking* service, which is provided by IMS Global to overcome resource linking challenges. Aside from CodeWorkout exercises, Mastery Grids launches other smart learning contents using the ADAPT2 protocol and the student progress is updated through the same protocol.

Mastery Grids accesses CodeWorkout exercises in a way similar to other available exercise types (i.e., loading the content inside an iFrame) but using LTI launch requests. As described in Section 4.3, the learning events generated at CodeWorkout (e.g., submitting a solution) are reported back to the Mastery Grids student modeling service through LTI grade callbacks. The Mastery Grids progress visualization is dynamically updated concurrently with the immediate feedback generated on the CodeWorkout exercise screen, as shown in Figure 12. Thus, the Mastery Grids progress indicator interface is currently working with multiple protocols, including the standard LTI protocol.

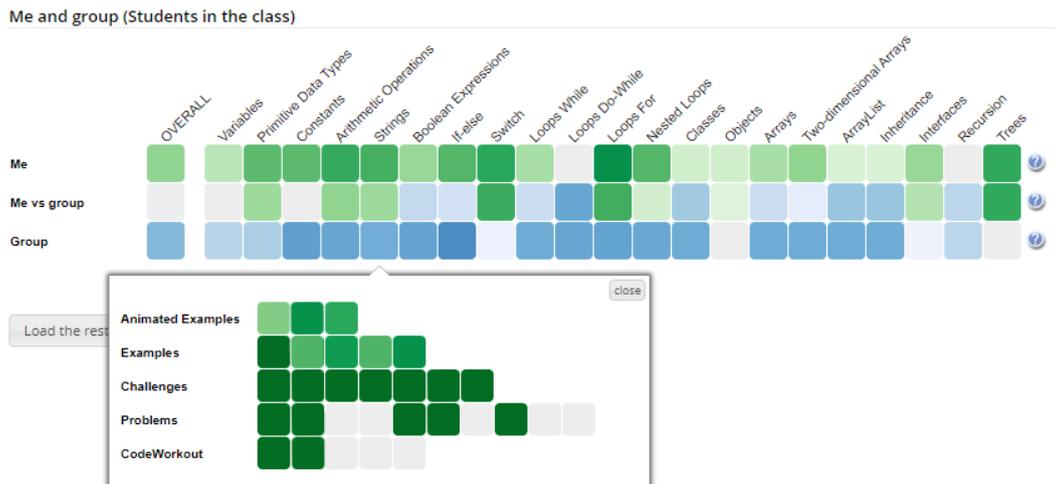


Figure 11. Mastery Grids interface prepared for a Java course. The available smart learning contents are shown for the topic Strings including integrated CodeWorkout coding exercises.

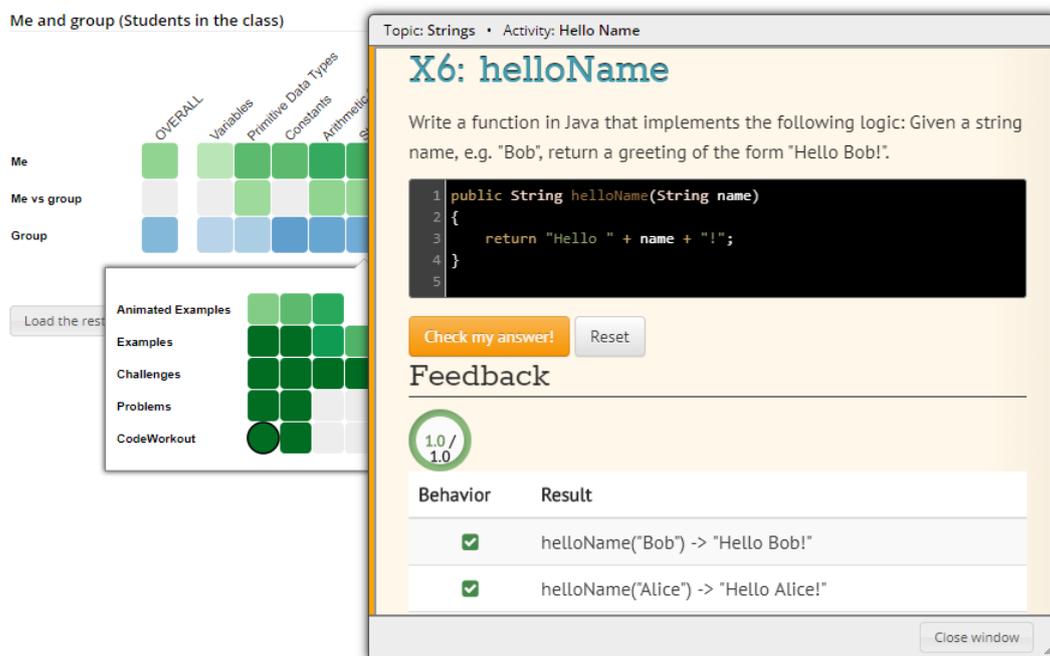


Figure 12. An instance of a CodeWorkout exercise accessed from the Mastery Grids interface. When a correct solution is submitted, the Mastery Grids progress visualization is updated (shown as a green circle).

5. Discussion and Conclusions

In this paper, we presented and explored a standards-based approach for broader re-use of smart learning content across multiple advanced content access tools. Our project was motivated by the need for content exchanged between two advanced tools, an online textbook system (OpenDSA) and a personalized content access interface (Mastery Grids). Each system was a centrepiece of its own infrastructure, providing access to various kinds of smart learning content. While the content provided by OpenDSA and Mastery Grids was complementary, there was no possibility to exchange the content across the systems due to the use of proprietary integration approaches. To make our infrastructures open and transparent for multiple kinds of learning content, we explored use of LTI protocol. While designed for a different context—integrating various tools into learning management systems—LTI offered a common ground for connecting different integration architectures.

By leveraging LTI communication, we implemented a standards-based approach to broadly re-use smart content between OpenDSA and Mastery Grids.

We have extended OpenDSA to support Python programming exercises and animations. OpenDSA had Java programming exercises and animations for various computer science topics for CS2 and above. It took years of effort by many collaborators across multiple institutions to develop those animations and exercises. Using the new infrastructure made it immediately possible to support Python textbooks using various Python exercises and animations from the ACOS server. It also opened the way to support a Java-based CS1 course through ACOS-based Java animations. Similarly, Mastery Grids previously was not able to provide good support beyond CS1 due to the lack of more advanced exercises. Leveraging the LTI-based approach, we were able to re-use a large number of CodeWorkout programming problems for CS2 courses.

We believe that the approach presented in this paper will open a way to a much broader re-use of smart learning content across various advanced-content access systems. We implemented two proof-of-concept scenarios to connect two specific kinds of smart learning content with two host systems. Our goal is to encourage other developers of educational systems to make their tools more inter-operable. Our systems are open source and so publicly available and we encourage developers to make use of our methods. Our solution is easily generalizable and can be applied for any kind of Web-based learning content. Moreover, as the number of LTI-compatible platforms increases, the amount of necessary modifications to support smart content re-usability gradually decreases.

At the same time, it is important to note that the LTI-based solution is not ideal for highly interactive smart content. The LTI protocol reports the final result of user work, for example, correctness of a problem solution or a percentage of explored animation. While this approach is sufficient for simple cases of integrating smart content learning objects (such as programming exercises) with an LMS, more advanced architectures are needed to deliver detailed learner analytics and to provide communications within an ecosystem of collaborating tools. To support such cases, a combination of LTI with other standards such as xAPI or CALIPER is required. We plan to explore these combinations in future work.

This work was performed with NSF support in the context of the SPLICE project (<http://csssplice.org>). SPLICE is focused on developing technical infrastructure for Computer Science Education research. In the context of this project, we collaborate with a number of research teams who develop and use innovative learning tools. As our work progresses, we plan to expand on ways to inter-operate with other infrastructures and collections of smart content.

Author Contributions: Conceptualization, H.M., P.B. and C.A.S.; Funding acquisition, P.B. and C.A.S.; Methodology, H.M. and J.W.; Project administration, P.B. and C.A.S.; Software, H.M., K.A. and J.W.; Supervision, P.B. and C.A.S.; Writing—original draft, H.M., K.A. and C.A.S.; Writing—review & editing, H.M., K.A., J.W., P.B. and C.A.S.

Funding: This research was partially funded by National Science Foundation grants DLR-1740775 and DLR-1740765, and by Virginia Tech's Technology-enhanced learning and Online Strategies unit.

Acknowledgments: We would like to thank Ayaan Kazerouni for his help in setting up a private instance of CodeWorkout.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

LMS	Learning Management System
LTI	Learning Tools Interoperability
ACOS	Advanced Content Server

References

1. Parlante, N. CodingBat. 2015. Available online: <https://codingbat.com/java> (accessed on 18 July 2019).
2. Buffardi, K.; Edwards, S.H. Introducing CodeWorkout: An Adaptive and Social Learning Environment. In Proceedings of the 45th ACM Technical Symposium on Computer Science Education, Atlanta, GA, USA, 5–8 March 2014; ACM: New York, NY, USA, 2014; p. 724.
3. Parsons, D.; Haden, P. Parson’s programming puzzles: A fun and effective learning tool for first programming courses. In Proceedings of the 8th Australasian Conference on Computing Education, Hobart, Australia, 16–19 January 2006; Australian Computer Society, Inc.: Darlinghurst, Australia, 2006; Volume 52, pp. 157–163.
4. Malmi, L.; Karavirta, V.; Korhonen, A.; Nikander, J.; Seppälä, O.; Silvasti, P. Visual Algorithm Simulation Exercise System with Automatic Assessment: TRAKLA2. *Inform. Educ.* **2004**, *3*, 267–288.
5. Diehl, S. *Software Visualization*; Lecture Notes in Computer Science; Springer: Berlin, Germany, 2002; Volume 2269.
6. Fouh, E.; Akbar, M.; Shaffer, C.A. The Role of Visualization in Computer Science Education. *Comput. Sch.* **2012**, *29*, 95–117. [[CrossRef](#)]
7. Shaffer, C.A.; Cooper, M.L.; Alon, A.J.D.; Akbar, M.; Stewart, M.; Ponce, S.; Edwards, S.H. Algorithm Visualization: The State of the Field. *ACM Trans. Comput. Educ.* **2010**, *10*, 9. [[CrossRef](#)]
8. Severance, C.; Hanss, T.; Hardin, J. IMS Learning Tools Interoperability: Enabling a Mash-up Approach to Teaching and Learning Tools. *Technol. Instr. Cogn. Learn.* **2010**, *7*, 245–262.
9. Shaffer, C.A.; Karavirta, V.; Korhonen, A.; Naps, T.L. OpenDSA: Beginning a Community Active-ebook Project. In Proceedings of the 11th Koli Calling International Conference on Computing Education Research, Koli, Finland, 17–20 November 2011; ACM: New York, NY, USA, 2011; pp. 112–117.
10. Fouh, E.; Karavirta, V.; Breakiron, D.A.; Hamouda, S.; Hall, S.; Naps, T.L.; Shaffer, C.A. Design and Architecture of an Interactive eTextbook—The OpenDSA System. *Sci. Comput. Program.* **2014**, *88*, 22–40. [[CrossRef](#)]
11. Ericson, B.; Cohen, J.; Miller, B. Using and Customizing Open-Source Runestone Ebooks for Computer Science Classes. In Proceedings of the 50th ACM Technical Symposium on Computer Science Education, Minneapolis, MN, USA, 27 February–2 March 2019; ACM: New York, NY, USA, 2019; p. 1240. [[CrossRef](#)]
12. Mitrovic, A.; Martin, B. Evaluating the Effect of Open Student Models on Self-Assessment. *Int. J. Artif. Intell. Educ.* **2007**, *17*, 121–144.
13. Loboda, T.D.; Guerra, J.; Hosseini, R.; Brusilovsky, P. Mastery Grids: An Open Source Social Educational Progress Visualization. In Proceedings of the European Conference on Technology Enhanced Learning, Graz, Austria, 16–19 September 2014; Springer: Berlin, Germany, 2014; pp. 235–248.
14. Sirkiä, T.; Haaranen, L. Improving Online Learning Activity Interoperability with Acos Server. *Softw. Pract. Exp.* **2017**, *47*, 1657–1676. [[CrossRef](#)]
15. Brusilovsky, P.; Malmi, L.; Hosseini, R.; Guerra, J.; Sirkiä, T.; Pollari-Malmi, K. An integrated practice system for learning programming in Python: design and evaluation. *Res. Pract. Technol. Enhanc. Learn.* **2018**, *13*, 18. [[CrossRef](#)] [[PubMed](#)]
16. Karavirta, V.; Ithantola, P.; Koskinen, T. Service-oriented approach to improve interoperability of e-learning systems. In Proceedings of the 2013 IEEE 13th International Conference on Advanced Learning Technologies, Beijing, China, 15–18 July 2013; IEEE: Piscataway, NJ, USA, 2013, pp. 341–345.
17. Brusilovsky, P. KnowledgeTree: A distributed architecture for adaptive e-learning. In Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers & Posters, New York, NY, USA, 19–21 May 2004; ACM: New York, NY, USA, 2004, pp. 104–113.
18. Sirkiä, T. Jsvee & Kelmu: Creating and tailoring program animations for computing education. *J. Softw. Evolut. Proc.* **2018**, *30*, e1924.
19. Ithantola, P.; Karavirta, V. Two-dimensional parson’s puzzles: The concept, tools, and first observations. *J. Inform. Technol. Educ.* **2011**, *10*, 119–132. [[CrossRef](#)]
20. Weber, G.; Brusilovsky, P. ELM-ART: An adaptive versatile system for Web-based instruction. *Int. J. Artif. Intell. Educ.* **2001**, *12*, 351–384.
21. Rey-López, M.; Brusilovsky, P.; Meccawy, M.; Díaz-Redondo, R.; Fernández-Vilas, A.; Ashman, H. Resolving the problem of intelligent learning content in learning management systems. *Int. J. E-Learn.* **2008**, *7*, 363–381.

22. Brusilovsky, P.; Edwards, S.; Kumar, A.; Malmi, L.; Benotti, L.; Buck, D.; Ihantola, P.; Prince, R.; Sirkiä, T.; Sosnovsky, S.; others. Increasing adoption of smart learning content for computer science education. In Proceedings of the Working Group Reports of the 2014 on Innovation & Technology in Computer Science Education Conference, Uppsala, Sweden, 23–25 June 2014; ACM: New York, NY, USA, 2014, pp. 31–57.
23. Hosseini, R.; Akhuseyinoglu, K.; Petersen, A.; Schunn, C.D.; Brusilovsky, P. PCEX: Interactive Program Construction Examples for Learning Programming. In Proceedings of the 18th Koli Calling International Conference on Computing Education Research, Koli, Finland, 22–25 November 2018; ACM: New York, NY, USA, 2018, p. 5.
24. Hsiao, I.H.; Sosnovsky, S.; Brusilovsky, P. Guiding students to the right questions: Adaptive navigation support in an E-Learning system for Java programming. *J. Comput. Assist. Learn.* **2010**, *26*, 270–283. [[CrossRef](#)]



© 2019 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).