Increasing Adoption of Smart Learning Content for Computer Science Education

Peter Brusilovsky
(co-chair)
University of Pittsburgh
peterb@pitt.edu

Stephen Edwards
(co-chair)
Virginia Tech
edwards@cs.vt.edu

Amruth Kumar (co-chair)
Ramapo College
amruth@ramapo.edu

Lauri Malmi (co-chair)
Aalto University
lauri.malmi@aalto.fi

Luciana Benotti
University of Cordoba
benotti@famaf.unc.edu.ar

Duane Buck
Otterbein University
dbuck@otterbein.edu

Petri Ihantola
Tampere University of Technology
petri.ihantola@tut.fi

Rikki Prince
University of Southampton
rfp@ecs.soton.ac.uk

Teemu Sirkiä
Aalto University
teemu.sirkia@aalto.fi

Sergey Sosnovsky
German Research Center for Artificial Intelligence
sergey.sosnovsky@dfki.de

Jaime Urquiza
Rey Juan Carlos University
jaime.urquiza@urjc.es

Arto Vihavainen
University of Helsinki
avihavai@cs.helsinki.fi

Michael Wollowski
Rose-Hulman Institute of Technology
wollowsk@rose-hulman.edu

ABSTRACT

Computer science educators are increasingly using interactive learning content to enrich and enhance the pedagogy of their courses. A plethora of such learning content, specifically designed for computer science education, such as visualization, simulation, and web-based environments for learning programming, are now available for various courses. We call such content *smart learning content*. However, such learning content is seldom used outside its host site despite the benefits it could offer to learners everywhere. In this paper, we investigate the factors that impede dissemination of such content among the wider computer science education community. To accomplish this we surveyed educators, existing tools and recent research literature to identify the current state of the art and analyzed the characteristics of a large number of smart learning content examples along canonical dimensions. In our analysis we focused on examining the technical issues that must be resolved to support finding, integrating and customizing smart learning content in computer science courses. Finally, we propose a new architecture for hosting, integrating and disseminating smart learning content and discuss how it could be implemented based on existing protocols and standards.

ITiCSE-WGR'14, June 23–25, 2014, Uppsala, Sweden

`http://dx.doi.org/10.1145/2713609.2713611`

Categories and Subject Descriptors

K.3 [**Computers and Education**]: Computer and Information Science Education; K.3 [**Computers and Education**]: Computer Uses in Education

General Terms

Design

Keywords

smart learning content, educational tools, classroom management, teaching with technology, intelligent tutoring systems, technology integration, dissemination, technology adoption, educational research, computer science education

1. INTRODUCTION

Education in the 21st century is being transformed by the Web. The Web now offers facilities that have the potential to rapidly change education at all levels. Blended learning, which augments traditional classroom education with online resources is now state of practice almost everywhere. Online learning resources such as offered by Khan academy (`https://www.khanacademy.org`) have empowered the individual to be an independent self-learner. At the

mass-education level, online technologies have made possible massive open online courses (MOOCs) [61] whose full potential is yet to be realized.

The embrace of online technologies by educators has resulted in the availability of not only a large number of learning resources, but also resources of high quality. In the past, text, images and videos have been the main forms of electronic content for a long time, often organized as course material in learning management systems (LMSs) such as Moodle (`https://moodle.org`). Course pedagogy has been augmented by various interactive services, such as discussion forums, chats and wikis, integrated into LMSs to provide communication and collaboration facilities between individual learners, and learners and teachers.

A more powerful breed of online resources has emerged during the past 10+ years, consisting of tools that have gone well past static content to provide interactive engagement from various forms of feedback to adaptation such as learner-controlled animation, dynamic visualization, and learner-led simulation. Typical examples within the computer science education domain include program and algorithm visualization and simulation tools [81, 78, 87], automatic assessment services for programming exercises [38], intelligent tutoring systems (e.g., [68]) and various forms of recommendation and social navigation tools, e.g.,[26].

In this report, we will focus specifically on such advanced content, which we call *smart learning content*. We will characterize its nature, and look at it from the points of view of different stakeholders such as students, teachers, content authors, tool developers, and administrators. Apart from adaptive personalization and sophisticated forms of feedback, smart learning content often also authenticates the user, models the learner, aggregates data, and supports learning analytics. We will examine the implications of providing these features on the technology needed to implement and support smart learning content.

We will analyze the challenges associated with using, authoring, and developing smart learning content, as well as related software support. We will discuss problems that inhibit adopting, sharing and customizing such content, and propose new technical solutions to overcome the challenges.

Our motivation for this work grows from multiple observations: 1) while many educators and researchers are developing smart learning content, most of the content has seen very limited use outside its home site; 2) it is often hard to integrate smart learning content with other smart learning content and/or LMSs; and 3) it is difficult to customize smart learning content to local needs. These problems have been recognized and documented earlier in several works. Roessling, Malmi et al. [72] discussed different kinds of computer augmented learning management systems (CALMS), a term they used for LMSs that are specifically designed for computer science education. They surveyed various specialized software tools for CS education, as well as pedagogical, practical and technical aspects of building CALMS. Roessling, Crescenzi et al. [73] took a narrower point of view and focused on how such CS specific learning resources could be integrated into Moodle. Recently, Korhonen, Naps et. al. [55] discussed similar issues in the context of interactive computer science education books that they called icseBooks. They designed a software architecture for the implementation of such books, taking into account the needs of various stakeholders (authors, teachers, and CSE researchers).

Our work in this paper continues along the above lines. We present a conceptual framework to analyze different kinds of smart learning content, and apply it to survey the current state of the art in the field. We analyze the challenges of adoption and interoperability of smart learning content from the perspective of different stakeholders (teachers, content authors, and tool developers) while also considering user modelling and personalization issues in smart learning content. We propose a new software architecture for integrating smart learning content, and describe how it could be implemented based on current protocols and standards in learning technologies.

Our report is compiled as follows. In Section 2, we will analyze in detail the different types of electronic learning content and formulate a definition for smart learning content. In Section 3, we will present an overview of the state of the art by summarizing our findings from online and literature surveys. In Section 4, we will analyze the problems and challenges in adopting and using smart learning content. Section 5 includes our proposal for addressing the main technical challenges. We will discuss the pros and cons of the proposal in Section 6. Finally, we contrast our contribution with related work and discuss future work in Section 7.

We acknowledge that readers of this report have different backgrounds and foci of interest, and we therefore give the following recommendation for reading the report. Those who are mainly interested in getting an overview of smart learning content and possibly adopting them as such, we encourage reading Sections 2, 3, and 6, as well as Appendix A. The readers who have a technical interest in integrating smart learning content, either as developers of such content themselves or integrating the content with other systems they are using, we recommend especially Sections 2, 3.4, 3.5, 4, 5 and 6.

2. DEFINING SMART LEARNING CONTENT

The term "smart content" has been used in various contexts, including the financial industry, business-to-business solutions and digital marketing [51]. In this report, to avoid confusion, we use the term smart learning content (SLC). In order to be able to define this term, we will first discuss the types of electronic learning content (ELC) currently available. Note that we will not include non-electronic learning content, such as printed textbooks or physical laboratory equipment in our discussion.

Traditional ELC includes static text, images and videos on web pages, typically organized as hypertext, which links various pages together. Wikipedia and YouTube are typical examples of such content. Such ELC has been usually organized for educational purposes using *learning management systems* (LMS), which can directly host content and provide links to external content. ELCs that incorporate interactive activities have included multiple choice questionnaires (MCQs) used for self-tests or quizzes.

While these ELCs have found use in all disciplines, in some disciplines such as computer science, educators have developed more elaborate learning resources that present dynamic and typically interactive learning content tailored to the discipline. For example, in the field of computer science education we can distinguish the following kinds of SLC (see See Appendix A for more information about some of the listed SLC):

- Program visualization (e.g., Jeliot, jGRASP, BlueJ [4, 19, 53]) and algorithm visualization (e.g., Animal or jHAVE [74, 65]) tools that allow students to explore and understand the dynamic nature of program execution and how algorithms work.

- Automatic assessment tools (e.g., BOSS, CourseMarker, Web-CAT, and Test My Code [32, 45, 24, 90]) that manage and grade student programming assignments submissions.

- Coding tools (e.g., CodeLab (`http://turingscraft.com/`), CloudCoder (`http://cloudcoder.org/`), and Codecademy (`http://www.codecademy.com/`) that support the learning of programming by having the student write snippets of code.

- Algorithm and program simulation tools (e.g., TRAKLA2 [54], JSAV [48], UUhistle [82]) that let the user carry out simulation operations using tailored graphical user interfaces.

- Problem-solving support tools (e.g., Problets [57, 58], js-parsons [40, 39], WadeIn [11]) that have the student learn concepts by solving targeted problems. Many of these are adaptive (e.g., Problets [56], ELM-ART [12], JavaGuide [36]) and can be classified as intelligent tutoring systems.

- Other kinds of systems such as social navigation systems (e.g., Educo [59] or Progressor [35]) that provide information about how peers have used and progressed through the learning resources, and thus support reflection of one's own working and progress.

In order to build a big picture of such a multitude of tools and systems and how they are related to each other, as well as to traditional ELCs, we need to identify different characteristics of these SLCs. Here, we attempt a canonical description of their essential characteristics. We start by positing that **some form of interactivity is a central aspect in all SLCs**. Content that is presented to learners without any interaction is not smart learning content. For the purposes of our definition, interaction denotes exchange of data between the learner and the learning content and happens in the following three stages (see Figure 1).

1. **Input** is the data provided by the learner to the SLC. It is on a continuum from *pre-specified* (e.g., play-pause-rewind buttons for animation and visualization) to *free-form* (e.g., code written by the learner, free-form input provided by the learner for visualization).

2. **Process** denotes how the SLC processes the input provided by the learner to select/generate output. If the SLC processes the input and generates/selects the output all on its own, it is *fully computational*. If on the other hand, the SLC merely facilitates the exchange of data between two or more human agents, it is *not computational*. The level of computational support is also a continuum from fully to none.

3. **Output** is the data provided by the SLC to the learner in response to the input. It is also on a continuum from *generic*, i.e., the same output is provided regardless of the learner/context (e.g., the same canned response to all learners: "thank you for your submission") to *customized*, i.e., the output is customized to the learner and/or context (e.g., whether the learner's answer is correct or not, the pages the learner must visit next based on prior performance of the learner).

Note that output entails the traditional notion of feedback, and customized output is what is traditionally referred to as adaptive feedback.

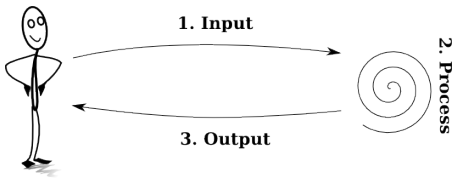

Figure 1: The three stages of exchanging data between the learner and SLC

The level of smartness of learning content can now be characterized as being on a continuum from pre-specified input/generic output to free-form input/customized output:

- A non-computational tool with pre-specified input and generic output is not smart, e.g., a traditional multiple-choice questionnaire (MCQ).

- A fully computational tool with free-form input, customized output is certainly smart, e.g., an automatic essay evaluation system which gives tailored feedback on various aspects of a submitted essay.

The continuum described above can be presented as a three dimensional space with axes along *Input dimension, Output dimension* and *Process dimension*. In Figure 2 we illustrate the categorization of various online learning tools along these three dimensions or axes. To avoid confusion in presenting 3-dimensional data, we present the Process dimension in terms of two adjacent pictures where the left one describes non-computational content and the right one fully computational content. Most examples fall somewhere between these extremes. Note that the list of tools categorized in the figure is not meant to be comprehensive.

In Figure 2:

- Hypertext is categorized as pre-specified input (learner's choices of hyperlinks are pre-specified) and generic output (pages do not change to suit the learner's needs). Adaptive hypertext also has pre-specified input, but the output is more customized than in the case of hypertext, because the selection and presentation of pages is based on learner interaction.

- Online MCQ without feedback is non-computational, has pre-specified input, and generic output (such as "thank you, your responses were received"). Online MCQ that provides summative feedback is computational and more customized. Online MCQ that provides formative feedback and allows the learner to revise his/her answer is more computational with more customized output.

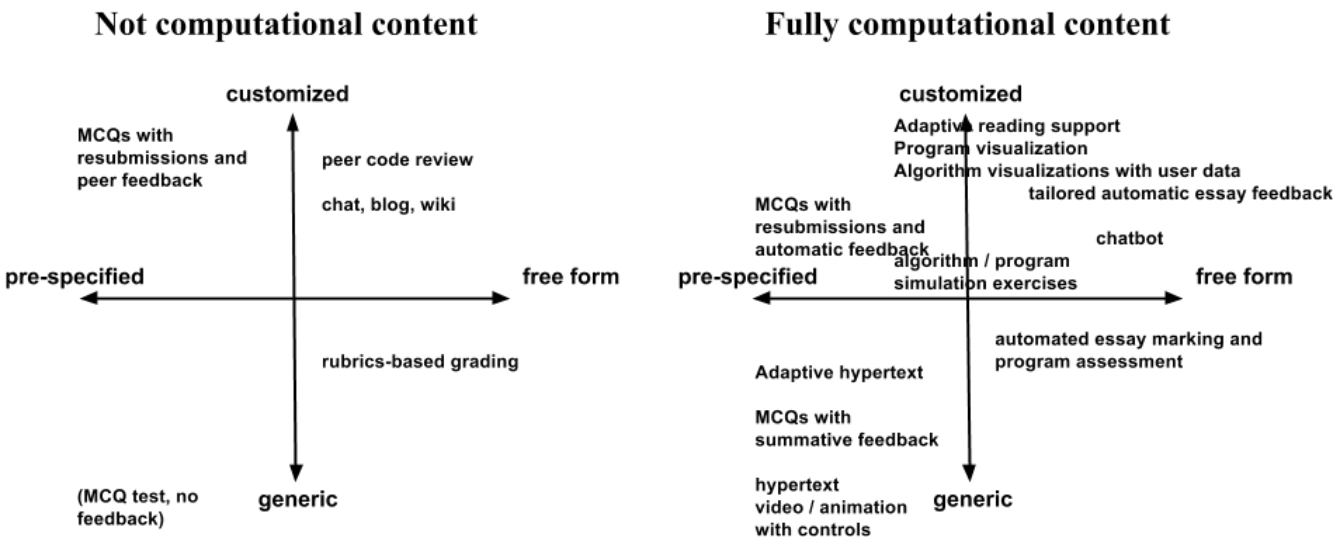

Figure 2: Examples of learning content classified by using input (horizontal axis), process and output (vertical axis) dimensions. Note, the left and right figures present the two extremes of the process dimension. For clarity, other values on the process dimension are not visualized. Note also that the items are simple some selected examples to highlight to categorization system, and their positions in the graph only approximately reflect their value within the two other axes

.

- Automated program assessment is fully computational, with free-form input (code), and somewhat customized output (based on the code written by the learner). Summative (only whether the program is correct or not) and formative feedback (feedback including why the program is incorrect) are on different parts of the output continuum.
- Algorithm visualization is fully computational (unless made manually by the user). Its input could range from fixed input data manipulation to free-form user input data, the latter including manipulation of the underlying code itself. The same applies to program visualization.
- Algorithm and program simulation exercises are computational (supported by the tool, although the learner manipulates the provided visualization). The input is pre-specified and the output is moderately customized.

Finally, in order to provide customized output, SLCs must collect data about the learner's interactions. They may **save** the collected data to display the learner's progress, provide suspend-and-resume capability to the learner, generate reports for the teacher and provide analytic capabilities to the author of the SLC.

3. SURVEY OF THE FIELD

This section provides an overview of the current state of the use and development of SLCs for computer science. It is based on an online survey of computer science educators asking them to identify the SLCs they use, and their opinions about the barriers to using SLCs (The survey instrument is included in Appendix C). To augment the collection of practically used SLC contributed by the respondents with most recent developments, we also surveyed recent literature. This survey produced an additional set of recently reported SLC. The collected SLC were reviewed and used to build a pedagogical categorization of SLC, which was, in turn, used to categorize the systems collected through both surveys[1]. In addition, we used the collected SLC as a basis to survey two important SLC issues: hosting solutions and data usage.

The section is structured as follows. The method for conducting the surveys is outlined in Section 3.1, while Section 3.2 discusses the responses from the online survey, outlining the use and challenges related to current SLCs. Section 3.3 provides a categorization of the SLCs from the surveys based on the dimensions in Section 2, and, in addition, discusses pedagogical foundations in current SLCs. Finally, Sections 3.4 and 3.5 discuss hosting smart learning content and gathering data from such systems respectively. The last two subsections may be of interest to only for those who are themselves engaged with developing smart learning content or integrating such content with other tools.

3.1 Method

The online survey was designed to gather information from the following aspects:

- What kinds of SLCs are being used by computing educators?
- What issues do teachers and students face in adopting and using SLCs?

[1]Appendix B provides the full list of SLC collected through both surveys classified according to the dimensions of smart learning content described in Section 2 as well as their pedagogical foundations discussed in Section 3.3.

- What needs and visions do teachers and students have when using SLCs?

The survey contained two parts: 1) Using smart content, and 2) Visions and challenges. To structure both parts of the survey we used simple categorization of SLC introduced at the beginning of 2. The complete set of questions is provided in Appendix C. The survey was sent out to the mailing lists of SIGCSE, csed-research, PPIG and Problets, as well as to a number of other lists as well as known affiliates and colleagues of the working group members. The SIGCSE mailing list has over 1200 members, while the Problets mailing list has over 400 members. Over a period of 10 days, 50 people answered the survey.

As the majority of systems gathered in the online survey were time-tested systems that had been in use for multiple years, the results of the online survey were complemented with a review of the last two years of ITiCSE to provide a view on recent advances in the development of SLC that have not yet been adopted by the wider audience. We acknowledge that a more complete literature survey would have been a valuable contribution, but carrying out such an analysis was unfortunately out scope of our resources, because the number of technical tools published in, say last 10 years is considerably large.

During the survey, in addition to outlining the state of the art of smart learning content, more technical issues in hosting and adapting existing systems were gathered and discussed by the working group members. The main themes of the issues are hosting and interoperability, and data collection and privacy. These themes are discussed in Sections 3.4 and3.5 that are written for more technically-oriented readers.

3.2 Nature and Use of Existing Smart Learning Content

Figure 3 illustrates the responses for the types of CS-specific SLCs educators had used since 2010:

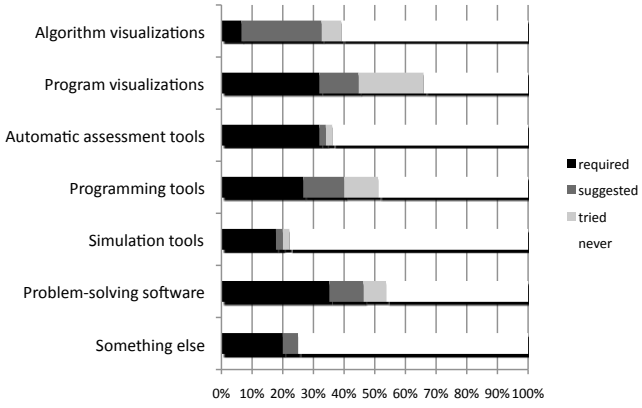

Figure 3: How different types of smart learning content are used by CS educators in their courses since 2010. *Required* label stands for teachers requiring their students to use SLC, *suggested* for SLC being encouraged to be used, *tried* indicates that a teacher has tried the corresponding SLC type by himself and *never* that a teacher has not tried anything like that before.

Majority of the respondents have used smart learning content, e.g., tried them, used them in their courses in some form, or suggested their students to use them. The survey indicates that problem solving software (e.g., Problets [57, 58] and WadeIn [11]) as well as program visualization tools (e.g., BlueJ [53] and jGRASP [20]) are the most used types of smart learning content, with almost 50% of the educators requiring or suggesting these tools in their courses. The least used smart learning content are simulation tools (e.g., TRAKLA2 [54] and UUhistle [82]). We acknowledge, however, that these results may be somewhat skewed due to a low number of responses from a specific community; in the survey, 51% of the respondents mentioned that they or someone at their institution had developed at least parts of a SLC.

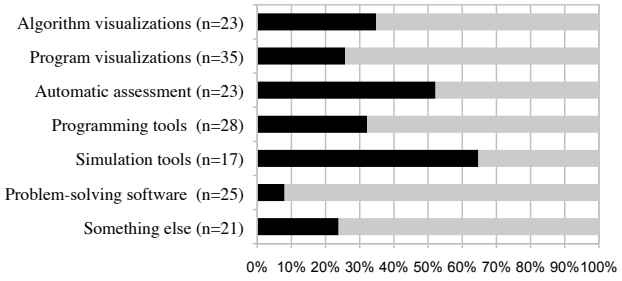

Figure 4: SLCs developed locally versus elsewhere. Black area indicates the portion of used SLCs developed locally, while the gray area indicates the portion of used SLCs developed elsewhere.

Figure 4 outlines tool types that have been most broadly adopted by other educators as well as tools that has been mostly developed and used locally. The SLCs that are most frequently used by educators but developed elsewhere include problem solving software, programming tools and program visualizations tools, while simulation and automated assessment tools were mostly used and developed in the same place.

When asking for the kinds of SLCs that educators would like to use in their courses, the most often listed SLCs were:

- Data and algorithm visualization, visual algorithm simulation for the languages Java and Python, as well as line by line visualization (also known as code animation).
- Automatic assessment and automated grading as well as self assessment for students.
- Interactive drill-like problems or small coding problems that can accept different answers.
- Peer programming and peer evaluation, for example, collaborative programming through collaborative editors.

The respondents also often mentioned that they would like to see additional features in existing SLCs (e.g., larger programs supported by CodingBat (`http://codingbat.com/`)), and SLCs that adapt to the user's knowledge or skill level.

In addition to the kinds of SLCs and SLC features that instructors use or would like to use, the survey asked for

reasons for using different kinds of smart learning content in instructors' courses. The more often mentioned purposes included:

- Giving students the chance to organize their self-study time.
- Flipping the classroom, i.e., the SLC would be used by the students to prepare for the class, or to prepare for a big closed lab project.
- Automated assessment and grading which would let students submit their projects multiple times, learn while debugging their code and improve their grade. This was thought to be a fairer evaluation mechanism in courses with hundreds of students where the learners get evaluated by different TAs.
- Live-demonstrations during lecture.

To consider the current adoption and challenges and difficulties in adopting SLCs, the survey participants were asked to select challenges that they faced from a list. The responses are summarized in Figure 5.

Figure 5 shows that the top three difficulties were that (1) respondents had problems finding SLCs that they could use; (2) they couldn't customize SLCs to fit their local needs and (3) they couldn't integrate the SLCs with other systems in their institution (e.g., to store results in grade roster). In the free-form category "Other", the three most often mentioned difficulties were usability issues, the time required to learn how to appropriately use the SLC and the unwillingness to use an SLC until its usefulness had been demonstrated.

Finally, participants were asked whether they had used data collected by the SLCs, and if so, the purposes for which they had used the data. Of the 34 who responded, 11 had never used any data collected by SLCs – one used external assessment to evaluate the effectiveness of an SLC whereas another was pleasantly surprised at the prospect of being able to use such data in the future. As to the purpose for which adopters of SLCs used the data collected by the SLCs, the top 3 purposes, according to the remaining 23 respondents were:

- Assessment of students in the courses—e.g., course/lab grade based on the data provided by the SLC: 8 respondents (35%);
- Assessment of the course—what works, what does not, and how the course could be improved: 7 respondents (30%);
- Research—developers for improvement of the SLC, and non-developers for longitudinal studies of their courses: 6 respondents (26%).

3.3 Categorization of SLCs

In the online survey, a total of 44 different systems were mentioned, while a survey over the last two years of ITiCSE mentioned an additional 17 systems. One of the systems (GreedEx [89]) appeared in both surveys and was included only once. A complete list of the 60 reported systems can be found in Appendix B. This section attempts to provide a brief analysis of these systems by categorizing then by their pedagogical foundation and the classification introduced in Section 2.

Pedagogical Foundations

When analyzing the systems mentioned both by the online survey respondents and the systems recently published at ITiCSE, a number of pedagogical foundations for the systems were identified.

- Active learning [6] is the foundation of SLCs that engage students either in performing cognitive tasks, such as coding the solution to an exercise, or playing roles other than that of a student (also based on sociocultural theory [91]), e.g., being the teacher who has to assess fellow students' solutions to an exercise. The input to these systems is free-form.
- Collaborative learning [21] is the basis of SLCs that provide users with environments that facilitate building collaborative solutions to problems. The input to these systems is free-form.
- External representations [76] are the basis of SLCs that present information to students in way that they can better build their own correct mental models. Since SLCs generate these external representations, they are "fully computational".
- Feedback [30] is provided by SLCs to help students achieve their learning goal. Since SLCs generate feedback, they are "fully computational".
- Individual differences theory [18] is the basis of adaptive systems that personalize content based on different criteria. With individual differences the instruction materials are adapted to students. Since SLCs perform the adaptation, they are "fully computational", with customized output.

We realized that Cognitive Load theory (CLT) [86] guided the design of other SLCs as well. CLT is devoted to optimizing students' mental performance. It differentiates among three types of cognitive load: intrinsic, extraneous and germane. Extraneous cognitive load is generated by the manner in which information is presented to learners and is under the control of instructional designers. We have not used in our classification because the border between being or not being based on CLT is not totally clear. However, we mention some systems that could represent the use of CLT, in addition to those classified within the External Representations category. On the one hand some SLCs provide interfaces that allow students to focus their attention on productive tasks, such as solving a problem, reflecting on the solutions given by others, etc. Some examples could be Cloudcoder (`http://cloudcoder.org`), BlueJ (`http://www.bluej.org`), Piazza (`https://piazza.com`) and DrJava (`http://www.drjava.org`). On the other hand, other SLCs organize the educational environment presenting topics with increasing complexity, e.g., DrRacket (`http://racket-lang.org`).

Table 1 summarizes the pedagogical foundations of the SLCs. The most common foundations are active learning, external representations and feedback:

- Active learning is present in a large number of SLCs (80.65%) including: algorithm and program visualization, simulation, automatic assessment, coding and problem solving.

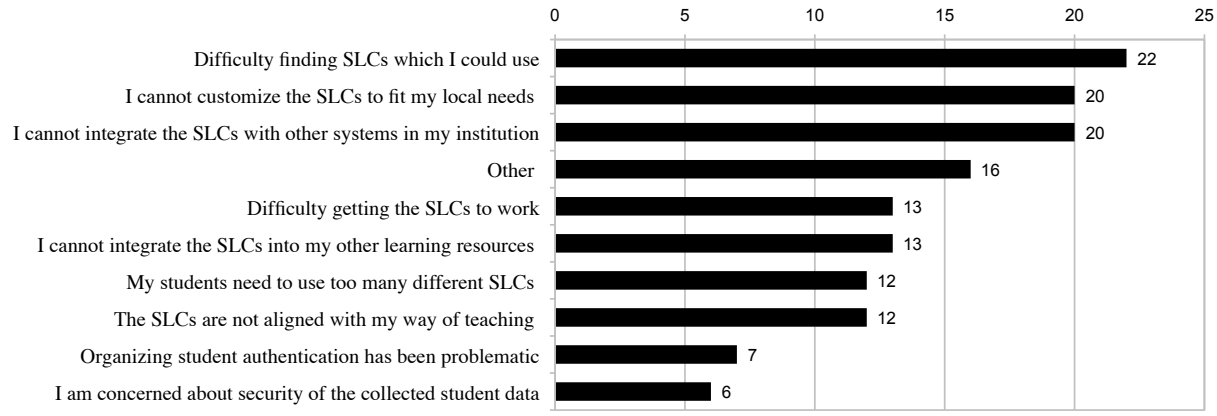

Figure 5: Difficulties and challenges adopting smart content reported by teachers sorted by how often they were reported.

- External representations are the basis of 53.33% of SLCs, most of them on algorithm and program visualization or simulation.
- Providing feedback to the users is the basis of at least 45.16% of the SLCs. Most automatic assessment, coding and problem-solving tools provide feedback.

Pedagogical aspects	# of SC	% of SC
Active learning	50	80.65%
External representations	32	53.33%
Feedback	28	45.16%
Individual differences	9	14.52%
Collaborative learning	4	6.45%

Table 1: Pedagogial foundations of the combined smart learning content

Dimensions of SLCs

All of the systems, both those suggested in the online survey and those found in recent literature, were categorised according to the dimensions described in Section 2. The intention of this is to give the reader a concrete understanding of the dimensions, as well as to analyse the distribution of current SLCs.

- 70.0% (42/60) of the SLCs are fully computational, accept free-form input and produce customized output. A majority of the systems are dedicated to algorithm and program visualization or simulation (e.g., [82, 89, 80, 3]), while the remaining SLCs are related to programming, problem solving and automatic assessment (e.g.,[24, 23, 22, 54, 93]).
- 11.7% (7/60) of the SLCs are not computational, accept free-form input and produce customized output, e.g., the Matrix simulation framework [47].
- 10.0% (6/60) of the SLCs are fully computational, accept pre-specified input and produce generic output. In this category, respondents often did not name concrete systems but described their use, which identified the systems as simple program visualizations and algorithm animations.
- 8.3% (5/60) of the SLCs are fully computational, accept pre-specified input, and produce customized output, e.g., Problets [57] and JSAV [48].

To provide an overview of the categorization, the results are also shown in Figure 6.

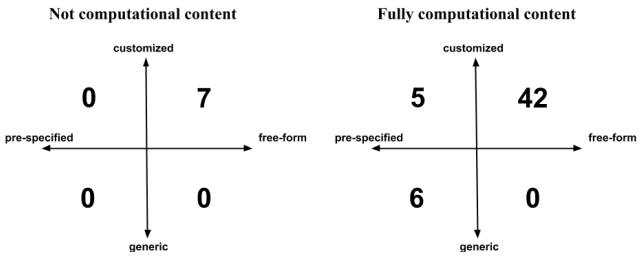

Figure 6: Combined Survey Classification Results

3.4 Current Architecture Solutions for Hosting

How an SLC is hosted can affect its adoptability and interoperability. The level of support provided by the delivery platform can vary widely in terms of where the SLC is fetched from and the kinds of SLCs and standards that are supported. We have identified five levels of host support based on the delivery platform's technical architecture and ability to serve SLCs:

At level 1 (the lowest level), the SLC is provided as a self-contained and independent service. The delivery platform is generic (e.g., the web) and does not integrate the SLC. No authentication is provided. So, any data collected during a session by the SLC is used by the SLC for personalization only during that session. The data is not stored between sessions to provide ongoing

personalization or tracking of the user. In this sense, this level is stateless.

At level 2 the delivery platform and the SLC are integrated into a single system. The delivery platform provides authentication and saves data produced by the SLC. Data can then be used across sessions to track the user and provide personalization across sessions. One of the limitations of this level is that the delivery platform does not support any external content—the SLC must be developed directly for the platform. Examples of the systems at this level are CloudCoder [34], Codecademy (`http://codecademy.com`), CodingBat (`http://codingbat.com`), CodeLab (`http://www.turingscraft.com`), TRAKLA2 [54] and Web-CAT [24, 23, 22].

At level 3 the delivery platform supports multiple SLCs, e.g., online MCQs along with program visualization. However, the platform does not support external content—all the content is internal or *native* to the platform. Examples of level 3 systems are ViLLE (`http://ville.cs.utu.fi`) and OpenDSA [28].

At level 4 , in addition to supporting multiple SLCs, the delivery platform allows the use of external content by supporting open but proprietary protocols. Examples include Test My Code [90], A+[46], JavaGuide [36], Database Exploratorium [14] based on the Adapt^2 open architecture [8], BlueJ or Moodle with plug-in support, and OpenEdX with its support for XBlock components.

At level 5 the delivery platform uses standard protocols (e.g., LTI) and hosts any SLC that uses standard protocols. Such a delivery platform is interchangeable with other such delivery platforms, and can interoperate with similar delivery platforms.

Both the delivery platform and SLC are weakly coupled—they use standard protocols (e.g., LTI), and an SLC can be hosted on different platforms and a platform can host different SLCs.

3.5 Data Collection and Usage

Most SLCs collect data. If stored, this data provides rich opportunities for the SLCs and delivery platforms to facilitate learning, teaching and research. This section describes the types of data traditionally collected by SLCs or their delivery platforms and outlines a set of purposes for collecting the data. It also briefly discusses the use of metadata to describe smart content, which facilitates discoverability, sharability, reuse and adoption of SLCs.

Collecting smart content usage data.

Data collected/stored by an SLC/delivery platform can be at different levels of granularity, listed in increasing order by size:

- Event-level: This data includes every mouse click, keystroke and operation of a GUI widget.
- Activity-level: This data includes details of the activity fostered by the SLC (e.g., solving a problem, visualizing an algorithm), such as 'problem started', 'problem attempted', 'answer submitted', and 'feedback read'.
- SLC-level: This data includes information about one session with an SLC, e.g., time spent using the SLC, aggregate performance on the SLC (number of problems solved, total score).
- Session-level: If the delivery platform is at levels 1 or 2 (described in Section 3.4), this is the same as SLC-level. If the delivery platform is at levels 3 or up, this data includes information about the various SLCs launched and used during the session.
- Student-level: This is the aggregate of all the data collected by the delivery platform for one student over multiple sessions, and includes additional information such as the name and demographics of the student.

Typically, the following relationship exists among the different levels of granularity of data:

$$\text{Activity-Level data} \supseteq \Sigma \text{ Event-level data}$$
$$\text{SLC-Level data} \supseteq \Sigma \text{ Activity-Level data}$$
$$\text{Session-Level data} \supseteq \Sigma \text{ SLC-level data}$$
$$\text{Student-Level data} \supseteq \Sigma \text{ Session-level data}$$

Event-level data is the most detailed, and student-level data is the most aggregate. Not every SLC and delivery platform may want to or may be capable of collecting data at all the above levels of granularity. SLCs should be able to collect data at least at the activity-level, but collecting event-level data within individual activities could improve the interpretation of learning outcomes. Analysis of the data must be carried out at the appropriate level of granularity:

- Event-level data can be used for data mining and learning analytics to infer learning-related parameters, patterns and episodes.
- Activity-level data is useful for item analysis, which is of interest to developers.
- SLC-level and session-level data is of interest to educators who want a report of class usage.
- Student-level data would constitute an electronic portfolio of the student's learning and would be of interest to both learners and teachers.

In a typical session, data at the above levels of granularity may be collected interspersed, as shown in Figure 7, where student-level (logging in), activity-level and event-level data are collected intermingled.

Delivery platforms at levels 2 and up (described in Section 3.4) save the collected data for reuse in future sessions. They may do so in one of two ways:

- Locally, using their own student model;
- On a third-party student model server, such as CUMULATE [13, 92] or Personis [49].

They may also archive the data for use by third-party learning analytics researchers. Some archiving options include TLA's Learning Record Store [60] and DataShop [52].

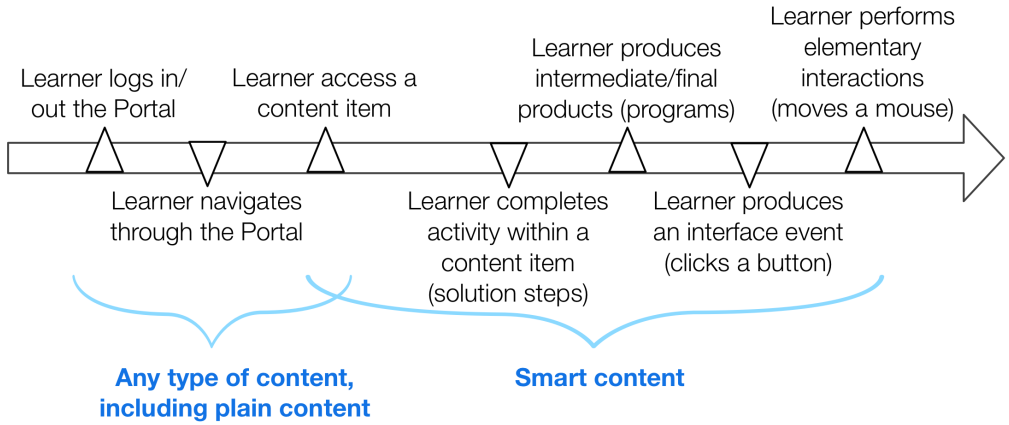

Figure 7: Content processing

Use of the data collected by SLCs.

The data collected by SLCs could be used for various purposes:

- By the instructor to 1) award students course credit for using the SLC; 2) learn about the concepts not well/yet understood by the students, and address them in subsequent classes; and 3) identify students in need of help.
- By the students to 1) track their progress; and 2) compare their performance against those of their peers.
- By authors of SLCs to 1) evaluate and improve the quality of the content and feedback presented by the SLC; and 2) evaluate the effectiveness of the SLC at helping students learn.
- By the SLCs themselves to 1) customize content and feedback to the needs of the learner; and 2) crowdsource generation of new content and effective feedback.
- By learning (analytics) researchers to 1) discover important learning episodes and phenomena (e.g., moment of learning, gaming the system, etc.); 2) get better understanding of learner characteristics; and 3) understand the features that make the SLCs effective.

It should be noted that aggregation of data from individual SLCs within a single data repository (either on the delivery platform or a third-party student model server) would provide added value to all concerned—teachers and students would receive more comprehensive reports, SLCs could better customize their content based on learner's interaction with other SLCs, and researchers would be able to use larger, cross-validated data sets with which to carry out data mining and learning analytic research.

Metadata to describe SLCs.

Metadata is used to promote integration and interoperability of SLCs, i.e., move delivery platform from level 1 towards level 5 (as described in Section 3.4). Typically, metadata includes descriptive data used for cataloguing (e.g., author, date of creation), pedagogical characteristics (e.g., difficulty, interactivity), and semantic information (e.g., the concept(s), skill(s) or competency(ies) the SLC helps to acquire). Metadata is often referred to as content model and does not exist as a separate document, but rather is built into the architecture of the SLC itself. Typically, it is open-access, machine-readable and standardized.

Metadata facilitates automatic discovery and integration of SLCs. Although current practices suffer from problems such as incompleteness and fragmentation of metadata standards, once these problems have been solved, metadata has the potential for increasing adoption and reuse of SLCs—delivery platforms will be able to automatically discover SLCs and offer them to learners when needed.

Availability of metadata is also crucial for customization of content presented to the learner—it helps the delivery platform decide how an SLC or the learner's interaction with it should be tailored to optimize the learning process.

4. INTEGRATING SLCS

There are many factors influencing the adoption of technology: technical, social, ease of use, and cost, to name a few. In this section we will focus on a problem that affects the adoption of SLCs even by their biggest proponents: the lack of integration of SLCs. The discussion in this section informs that of the next in which we propose a prototype architecture as well as that of the subsequent section in which we evaluate the proposal. As mentioned in the introduction, both sections target the readers who have a technical interest in developing and integrating smart learning content.

Respondents to our survey identified the following three difficulties of adoption of SLCs the most (see the Figure 5 in Section 3):

- Problems finding SLCs they could use;
- Customizing SLCs to fit their local needs; and
- Inability to integrate the SLC with other systems.

Some other notable responses were about usability—"If an SLC doesn't have an incredibly intuitive interface, it will not get adopted"—and the (mis)match between the SLC and teachers' needs—"Content in many SLC's don't match what/how we teach", and "nothing fits the pedagogy/curricular structure I have, so reuse is hard".

4.1 Technical Integration

Monolithic Systems

In the current landscape, an SLC is often a monolithic system. The challenge of integration begins with features as mundane as that of requiring separate logins. In some instances, this may require the instructors to create accounts for their students, posing a high barrier for adoption. Often, monolithic systems do not communicate back to the launching system (e.g., an LMS) or they do not communicate with each other. In some instances, an SLC is installed as a stand-alone desktop application.

A further challenge is the granularity of accessing an SLC. Often it is an all or nothing approach, one that does not allow the instructor to point directly to the desired materials. In this environment, integration many times takes the form of a simple table of contents linking to different SLC sites. While such an approach offers access to desired materials, it lacks finesse and makes it hard to configure a course to the instructor's needs and to personalize contents to a student's needs.

Our survey respondents expressed concern about the ease of management of student records. *"Student management for some of these programs is also a problem. Moodle allows for download of the student records so that management is easier. The csv file can be imported into excel and edited there." "The SLCs that I have used all require manual uploading of data into the course management system. This gets rough on the TAs when there are 150+ students in the course and this is not their only duty. Also, there are lots of problems with students doing stupid things (like not putting in their names–or using a nickname that is different than their real name). And very few of these systems have space for a student ID #, that would help to solve these problems."*

This addresses a third and perhaps most critical challenge for an integrated system: accessing data from all the SLCs in one place. This feature offers the following advantages:

- An efficient way to get basic feedback, such as assignment scores into the course gradebook.
- A more holistic and complete view of the student's progress.
- Combined logging of data that facilitates research about the usefulness of a particular tool in a course.

In regards to the last item, some teachers are not sure about the value of SLC. *"I would like to see statistically validated research that shows that SLCs work better than other approaches before I consider using them".* Hopefully, research on the effectiveness of SLCs will convince instructors of their usefulness.

Integrating SLCs

In the prior section, we argued in favor of integrating multiple SLCs within a single delivery platform. In this section, we will highlight the challenges in integrating SLCs.

There is currently no standard protocol for interacting with SLCs. This in itself tends to perpetuate the creation of monolithic systems. Furthermore, monolithic systems are typically produced and consumed locally. As such, they may not be designed to be integrated with other environments.

Issues of intellectual property rights and credits are straightforward in a monolithic system, but must be carefully addressed in a shared environment. In this context, if an SLC is not free, then piecemeal pricing at various levels of granularity should be made available. Then again, paying for an SLC may itself be the problem—to quote some of the respondents to our survey: *"My school doesn't want to pay for any SLCs" "Can no longer require students to pay for anything other than textbook so SLC must be free for them to use".* Monolithic systems use their own layout and terminology. The terminology may work well for their intended local use. However, it may introduce conflict or at least confusion when integrated with other SLCs. Differences in the user interfaces of different SLCs will likely make navigation hard and may additionally pose problems for students with particular learning disabilities. In general, the ability to configure the user interface of SLCs is desirable. If an instructor wishes to modify the content of an SLC, an appropriate authoring tool must be provided by the developer of the SLC, along with documentation on how to use it. The SLC would necessarily have to save and retrieve such customized content from a centralized repository. However, this would result in "crowd-sourcing" content, enriching the SLC, making its adoption more attractive to other users, and creating a community of users in the process, along with the motivational benefits that such communities afford. An instructor may wish to know about the usefulness or popularity of a given SLC. So, developing a social network of users, and posting ratings and learner feedback may all promote adoption and use.

Data and Its Collection

Finally, the question arises as to the ownership and use of data collected by an SLC. Any data that is made publicly available for use by third parties must be stripped of identifying information to stay in compliance with privacy laws and IRB policies. Nevertheless, there is a desire in learning research community to make data collected by SLCs available so that third parties can validate research results of the SLC and/or conduct their own research.

4.2 Conceptual Integration

Hurdles which potentially preclude the inclusion of a particular SLC into a course include:

- The use of different standards, e.g., some educators use only int and double data types whereas others embrace the whole range of data types in Java/C++
- The use of notations and terminologies, e.g., on balanced binary search trees, one and the same rotation may be called "single rotation on the left sub-tree" by some authors and "single right rotation" by others.
- Examples written in different programming languages, e.g., Java and C++.
- Components authored in a different natural languages, e.g, code animation tools written in English versus Spanish.

To streamline course pedagogy, educators may want to customize an SLC with the specific notation they use in class, so as to be consistent with their lectures and/or across

the SLCs they use in the course. They may also want to substitute local vernacular for concepts covered in the SLC (e.g., 'modulus' versus 'remainder' operator). SLCs should provide facilities for such customization, either through configuration variables or command-line arguments.

From a student's perspective, adaptation of an SLCs may mean catering to one or more of the following:

- A specific level of Bloom's taxonomy [5], e.g., synthesizing/writing a for loop versus analyzing the behavior of a for loop.
- A specific learning outcome.
- A specific learning style, e.g., verbal versus visual learners and sequential versus global learners in Felderman and Silverman Learning Style Inventory [27].

In order to facilitate such adaptation, either SLCs or the activities within them should be annotated with the level of Bloom's taxonomy, the specific learning style, content topics covered, etc.

The survey respondents noted that too much time was required to learn how to appropriately use an SLC: *"It takes too long to figure out how to use the SLCs"*, *"Time investment to learn and integrate SLC is too high"*, *"Find the time to learn the tools myself, and make appropriate adjustments to existing course"*, *"Just haven't had time to explore and learn and develop."* This is a very good argument in support of seamless integration of various SLCs wherein they all share the same user interface, and present the collected data using comparable ontology, granularity and format.

4.3 Delivery Integration

The advent of the Web simplified the delivery of SLCs—educators no longer had to download and install SLCs in order to use them. Networked languages such as Java simplified the production of SLCs—authors of SLCs could implement them once and have them run on a multitude of client hardware/software environments. Unfortunately, this SLC utopia lasted less than a decade—both production and use of SLCs are fragmenting again.

Users of SLCs are increasingly migrating from personal computers towards tablets and smartphones, which do not support the Write-Once-Run-Many models of personal computers (e.g., Java applets). The technology that runs on both personal computers and handhelds (HTML 5) is constantly evolving, miring authors in a continual cycle of rewriting their SLCs. The rapid change in the landscape of implementation of SLCs has meant many high quality legacy SLCs are falling into disuse prematurely. These developments pose a significant obstacle to integration, interoperability and reuse of SLCs, especially those developed by different authors. Unfortunately, this problem is not expected to find a resolution anytime soon, if at all.

5. A PROTOTYPE ARCHITECTURE FOR INTEGRATING SMART LEARNING CONTENT

5.1 Proposed Architecture: The Core

In this section, we propose an open architecture that supports integration of SLCs into delivery platforms. The proposal is based on our analysis of stakeholders' needs, existing solutions and best practices. It was designed with multiple objectives in mind:

- To lower adoption barriers for smart content by enabling teachers to integrate multiple SLCs within their courses in accordance with their preferred way of teaching.
- To provide for finer-level personalization by adapting the delivery and usage of smart content to the current knowledge and skills of individual students.
- To address the issues of content discoverability, and community-of-practice support.

The architecture is based on three principal ideas: flexible reuse of SLC, flexible course organization, and flexible data collection, which have been accepted as best practices in the field. This flexibility is supported by decomposing the critical functionalities that have to be implemented to support the use of SLC into three layers: the delivery layer, the application layer, and the data layer (Figure 8). The delivery layer is formed by various student-facing delivery platforms. The application layer is composed of multiple SLC servers that deliver various kinds of SLC. The data layer hosts all kinds of Learning Record Storage systems that collect and store information about students' work. The reasons for this decomposition, the nature of the component forming each layer, and the communication between these layers are explained below.

Flexible Reuse of Smart Learning Content

Flexible reuse of SLC is supported by the separation of delivery platforms and SLC servers, which breaks the currently predominant encapsulation of SLC in standalone systems. By delivery platforms, we mean course management systems, learning management systems, MOOC platforms, or other course delivery mechanisms that are intended to be the student-facing access points for course learning materials. Typical SLCs are implemented independent of such delivery platforms at present, and instead take the form of stand-alone systems that provide their own student-facing entry points or front pages.

To increase the reusability of SLC, it should not be encapsulated in a single stand-alone system, but instead should focus on supporting embedded use within existing delivery platforms–in effect, being hosted on independent *SLC servers*. Each SLC server should be able to host multiple SLCs and be able to deliver any item by request to a delivery platform. The separation of SLC servers from delivery platforms should be based on a standardized communication interface. The interface should allow any delivery platform to launch any individual SLC directly from any content server and receive back information about user's performance with the SLC.

More specifically, a delivery platform should be able to get sufficient information to launch an SLC (a reference to a specific SLC on a specific SLC server) while passing two essential parameters—the identity of the individual user and the identity of a specific LRS system. In addition, the communication protocol should allow a richer context in the form of name-value parameters to be passed from the delivery platform to the SLC, This information could be used to customize the delivered SLC (e.g., communication language) or passed to the LRS system for storage (e.g., information

about a class, a specific learning treatment, etc.). Once the user completed the work with SLC, the delivery platform should receive some some summative information about the results of this work that is most important for the delivery platform to operate: success, completion level, or performance score.

Flexible Course Authoring

Flexible authoring supports instructors in customizing SLC and organizing it in a way that fits the structure of their course or preferred way of teaching. We distinguish two levels of authoring support: content-level authoring and course-level authoring. *Content-level authoring* support allows instructors to extend the volume of SLC by adding new content, as well as cloning and editing existing items of an SLC. We expect that content-level authoring features are tightly coupled with the nature of a particular SLC, and thus will most likely be supported through an authoring interface hosted by individual SLC servers. *Course-level authoring*, on the other hand, should allow instructors to organize SLCs of multiple types that they want to use in their courses in a way that supports the unique needs of a course. Existing course delivery platforms most frequently use a common approach to organizing course materials, defining a course structure as a sequence of units (topics, lectures, course weeks) where each unit provides access to a collection (or a sequence) of educational content items of different kinds that support mastering this unit. This kind of course structuring can accommodate embedding SLC activities as long as any SLC from any SLC server could be added/inserted/embedded to any unit within a course structure. Given the popularity and the common acceptance of this approach, we suggest it as the main approach to flexible course-level authoring with SLC. The standardization of the proposed two-level course structures with smart content references could turn course structures into reusable assets making it easier to store and clone course structures as well as ensuring the ability to use the same course structure on multiple delivery platforms.

The proposed flexible course authoring could be easily supported by in any delivery platform that can reference SLC. At the same time, it could be wise to separate flexible course authoring from delivery platforms and provide a community-of-practice portal for course-level authoring. This platform could collect and propagate community wisdom in course organization. With a community-of-practice portal, building a new course structure from scratch will not be the only approach to create a course with SLC. Instead, one of the existing well-designed structures could be cloned and edited for further customization. For example, we envision a number of exemplary course structures complete with links to various SLCs that support a specific programming textbook. Faculty who choose to adopt this book for his or her course could also adopt the whole course structure and have a course with SLC ready for delivery with minimal efforts.

Finally, while we propose a lightweight standard for representing and sharing SLC-based course structures, it is clear that no single course authoring approach, even flexible, would satisfy all the teachers who are interested in using SLCs. For example, it is quite common nowadays to organize course content as an online book with smooth narration and links to interactive content embedded into the book structure [55]. However, the separation of SLCs from delivery platforms can benefit any course organization approach. An easy way to combine any desired course-level authoring approach with SLC reuse organization would be to create a proprietary delivery platform that supports SLC inclusion standards and allows publishing multiple courses in the same desired format. A good example of such platform is Open DSA [28], a platform that allows creating textbook-organized courses that embed SLCs. A more challenging approach is to create a specialized standard for the desired course organization (such as a textbook) and ensure that this standard is supported by several delivery platform just as a hierarchical folder-based standard organization is supported by almost any modern delivery platform.

Flexible Data Collection

Flexible data collection facilitates the collection of data about learners by SLCs. The majority of delivery platforms were created with plain static content in mind and were engineered to collect minimalistic information about user learning, i.e., the fact that a content item has been accessed and, for assessment items, the correctness and score. In contrast, smart content typically engages user in interactive work and can produce a much richer trace. This trace is important for learning analytics and personalization needs, but learning platforms have no space to store it. Moreover, for the case of smart learning content that could be used by many delivery platforms, encapsulating learner data inside specific learning platforms limits the ability to learn about the users and the content.

To address this problem, we propose independent learning record storage (LRS) systems to facilitate data collection by SLCs. The idea of an independent component for collecting user data has been already explored in the field of adaptive educational systems (where it is known as student model or user model servers) [92, 49] and learning analytics systems [88]. It has also been advocated by Advanced Distributed Learning group (ADL) as a part of their new Training and Learning architecture (TLA, `http://www.adlnet.gov/tla/`). LRS systems should support a standard data reporting protocol to collect data as well as standard data access protocol. Thereafter, any standards-compliant LRS system will work with any SLC or SLC server. A specific LRS server should be selected within the delivery platform. A delivery platform might suggest or mandate a default LRS system. However, the open architecture also allows a teacher to select the preferred LRS system for his or her class or allow students working with a course outside of any class to select a specific LRS system for aggregating all their learning data. In any case, the information of the selected LRS system should be passed by the delivery platform to each SLC or SLC server to ensure that all learner data flows to the same storage.

A smart content item should be able to send a rich flow of information with details about each critical pedagogical event to the selected LRS system. The information passed about each event should be standard by its form while being specific in its semantics for different types of smart content. It should be extended with context parameters passed from the delivery platform. The level of detail for reporting user experience could be decided by the smart content server, but it should be creating a sufficiently rich trace for the purpose of learning analytics, user modeling, or data mining. These kinds of systems can be considered as clients of the LRS

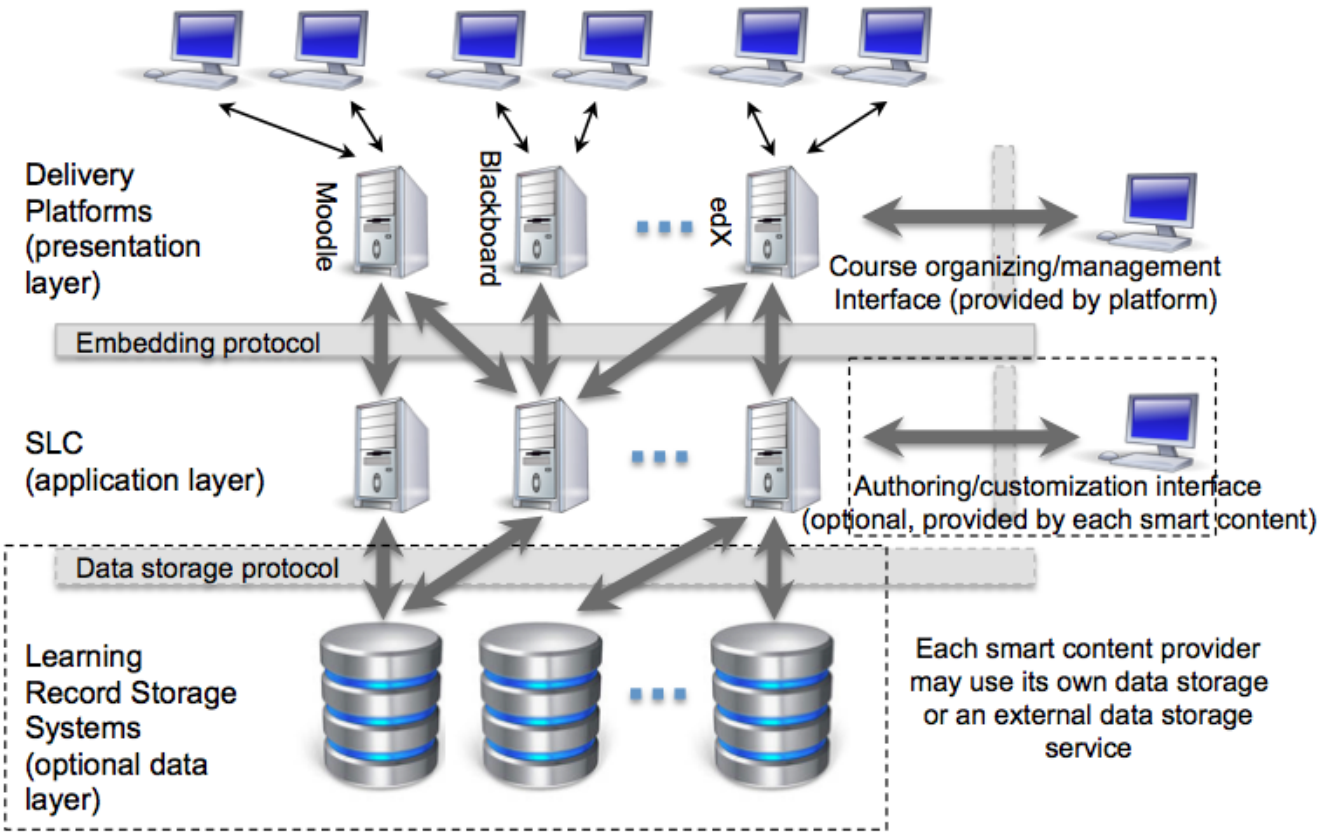

Figure 8: A three-tier architecture for integrating SLCs into delivery platforms.

system and should be able to request information from it using a query protocol. A critical part of client to LRS system communication is reliable access control that can ensure privacy of data.

5.2 Addressing Discoverability in the Proposed Architecture

Discoverability of SLCs emerged as the top concern in the online survey reported in Section 3. The discoverability issue has the same roots as the integration problem: due to the unique nature of SLC, individual SLC items (e.g., individual activities, lessons, or exercises within one SLC system) cannot be easily collected in learning repositories or indexed by search engines. As the review of the field shows, the majority of SLCs are standalone systems. As a result, searching the Web or browsing repositories at best leads one to an SLC as a whole, and not to its constituent activities or components. So, even after discovering a promising SLC, potential adopters have to invest considerable time exploring it and evaluating its appropriateness.

This suggests the need for SLC discovery tools that implement the SLC connection interfaces suggested for delivery platforms. A discovery tool of this kind would make all smart content items on all servers known to be discoverable to users through search and browsing, while also allowing users to immediately launch and try any discovered item.

To support such discovery tools, SLC authors can provide a mechanism for indexing every SLC item with both keywords and metadata, so that this information can be published in a machine-readable format. Next, the architecture can be extended with an additional standard communication interface that we call a *content brokering interface*. This interface would allow SLC discovery tools to both discover and request specific content on any SLC server—either a single item, or the full list from that SLC. Any learning repository that supports this interface will immediately get access to all content on any registered or discovered SLC or SLC server (see Figure 9). Finally, content brokering can also be embedded into course authoring tools to facilitate incorporating SLC items (and not just SLCs) into a course.

Since it might be hard to affect the process of course authoring on multiple delivery platforms, an easiest way to achieve this goal is to create a delivery-platform-independent "smart course" authoring tool that allow authors to search for proper smart content and embed it in parallel with defining course structure. Once the structure is created, it could be packaged into a standard format and uploaded to a delivery platform of choice. An authoring-focused component that integrates course structures from multiple authors regardless of the platform they use for delivery opens enhanced opportunities to support communities of practice and teacher-oriented personalization. An example of teacher-oriented personalization that could be embedded into an authoring tool is item recommendation based on collaborative

filtering. Collaborative filtering may identify relevant learning units in different courses on the basis of their usage in other courses and cross-recommend most successful and useful items.

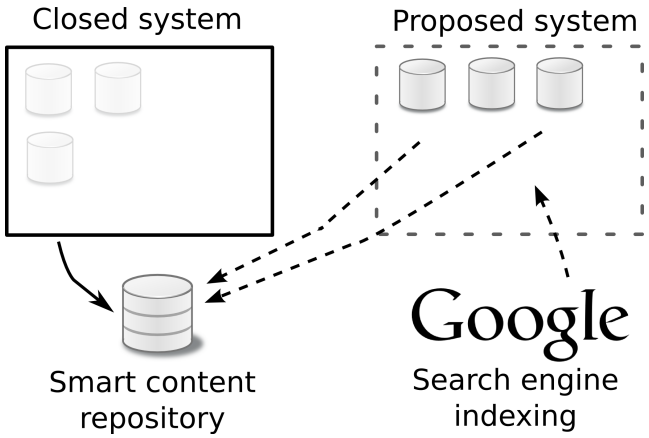

Figure 9: Our architecture makes access to smart content transparent for both discoverability and reuse.

5.3 Provision for Personalization Support

While personalized and adaptive learning has not been explicitly included in survey questions, respondents mentioned it among the desired features of SLCs. Provision for personalization has been embedded in the proposed architecture through the event-reporting interface. This arrangement ensures that critical information about learners' activities is collected, while being free from its usual captivity inside various delivery platforms and SLCs. Thus, learning records could be used by any component of the proposed architecture as long as access rights and privacy rules are observed. The information about student activities stored in an LRS system could be used to determine his or her current level of knowledge. In turn, this information could be used by any other component to adapt to individual users. For example, a delivery platform could use adaptive hypermedia technology [9] or collaborative filtering technology [77] to guide students to most appropriate SLC item. This personalization will break the static linear content browsing approach, thereby offering efficient personalized learning path through SLC content for each student. SLC servers could also use information about learner's knowledge to adapt the work of its SLCs to individual students, for example, offering adaptive program visualization [11].

Learner modeling is a challenging task that can be prohibitively expensive for SLC systems to implement from scratch. To avoid re-implementing this core feature in separate SLC systems, we recommend implementing learner modeling functionality as a separate component type in the architecture, which we call learner modeling server. This approach fits much better to the distributed nature of the proposed architecture and replicates a popular approach in the field of user modeling where a user modeling component or server is frequently separated from applications that rely on its functionality. As mentioned above, several *user modeling* servers such as Personis [49] and CUMULATE [92] have been predominantly used for learner modeling in an educational context. In the context of our architecture, the goal of a user modeling server is to process a flow of information about learner actions and achievements into a set of parameters that define current knowledge, interests, and behavior parameters. This information could be requested by any authorized component that needs to adapt to an individual student. Note that the architecture allows many learner modeling servers to work in parallel and compete in terms of reliability and quality of modeling. A component that wants to deliver personalization needs to pick a preferred server and to authorize it to access learner data.

The most challenging part of learner modeling is estimating a learner's level of knowledge. Some simple knowledge modeling could be implemented just by taking into account the amount of work and the context (i.e., course topic) passed to the LRS server. However, a more advanced model that is essential for finer-grained adaptation should be based on some competency model in the domain that is usually defined as a level of achievement in a taxonomy or ontology of skills and concepts. These kinds of ontologies have already been created and used for adaptation in the domains of C programming [84], Databases [83] and Java programming [10].

Competency-level adaptation requires any activity done by the user to be associated with ontology competencies. One way to assure this is to pass information about competencies applied or tested along with the pedagogical event record. An easier way, however, is to associate every SLC item with a set of competencies that the learner can practice or test while working in this item. In this case, the competencies associated with a SLC items could be stored as a part of content-level metadata and provided by the content server through content brokering interface by request. Merging this metadata information with the flow of events recorded by the LRS server, a learner modeling server could create a fine-grained picture of user knowledge. This approach has already been used by such servers as CUMULATE [92].

5.4 Pragmatic Vision: Using Existing Protocols and Standards to Provide a Near-term Solution for SLC Integration

The previous sections proposed an open architecture (Figure 8) based on standard protocols for communication between the delivery platform, SLC server, and an LRS system. It also discussed course re-use based on standardized course structures. This proposal was based on needs analysis while mostly ignoring existing standardization attempts. In this section, we propose an expedient architecture that uses existing protocols for communication between the delivery platform, SLC server and LRS. While this architecture may not meet all the needs of SLCs, it has the advantage of being immediately achievable, helping to meet current needs while researchers work toward a more ideal solution.

A Pragmatic Approach to Implementing the Embedding Protocol

If SLC developers want their work to be embeddable into existing delivery platforms, it is necessary to use existing embedding protocols or APIs that are supported by current technology. While many platforms support their own custom embedding approaches (e.g., Moodle's plugins, or edX's XBlocks), the most widely supported standard cur-

rently available for embedding learning resources in delivery platforms is the Learning Tools Interoperability standard (LTI v2.0, http://www.imsglobal.org/LTI/). Most major delivery platforms support the use of LTI-compliant tools, including Blackboard, Moodle, Canvas, Desire2Learn, Sakai and edX.

Because of the broad existing support for LTI by delivery platforms, it is an ideal target for near-term use by SLC developers until a more comprehensive embedding standard evolves. Unfortunately, providing an LTI-compliant interface for existing SLCs can be a significant challenge. The standard is detailed and includes coverage of many features that may not apply to specific SLCs. Implementing an LTI-compliant interface may be costly.

As an alternative, a promising near-term solution may be the implementation of an LTI-compliant "adapter" that provides a cleaner, lighter-weight API for communicating with the most common class of existing SLC services. Such an adapter could be used as a simple repackaging mechanism to add LTI-compliance to any SLC service, allowing it to be embedded in current delivery platforms that support LTI. Such an LTI-compliant adapter could dramatically reduce the cost of providing LTI support for SLC developers, while also opening up a wide variety of delivery and integration options with off-the-shelf software. We believe that this approach is the most promising near-term solution, until a more expansive embedding protocol or API can be developed that more completely meets the needs of SLC services.

A Pragmatic Approach to Implementing the Data Storage Protocol

In current practice, most existing SLCs provide their own data storage, using custom data models. For near-term use, continuing this practice is a viable choice.

At the same time, however, there is growing recognition that there are benefits to be gained by storing data in a separate (possibly shared) location. Started in the work on learner modeling and learning analytics, this approach has been popularized by ADL Experience API, or xAPI (http://www.adlnet.gov/tla/experience-api/, formerly known as the Tin Can API, http://tincanapi.com), which is a recently developed specification aimed at allowing SLCs to store data in a standardized format in a learning record store (LRS). One SLC may store data in multiple learning record stores, and a learning record store may receive data from multiple SLCs, and may even exchange data with other stores. xAPI allows independent development of learning analytics software or data analysis techniques that can be applied across data collected from multiple SLCs.

Although the xAPI is relatively new, client libraries are available for Javascript, Java, PHP, Objective-C, and .NET that developers can use to incorporate the protocol into their SLC (http://tincanapi.com/page-developers/). So, it is the most expedient option for SLC developers who are looking to migrate from self-contained data storage to using independent learning record storage services.

A Pragmatic Approach to Implementing Course Re-use Standards

While re-usability of SLC-based course structures has not been addressed by existing standards, re-usability of traditional courses has been extensively explored. As a results, a very small extension of existing tools and standards can resolve most of the problems in SLC course-level re-use. A possible solution could be to represent a link to an SLC item as a specialized URL leaving the proper resolution of the link to the delivery platform will enable the use of existing authoring tools and standards. For example, SCORM-compliant authoring tools originally aimed at helping educators package up course materials for transportability among learning management systems may be leveraged to create SLC-enabled courses. Moreover, content reusability standards such as SCORM (http://www.adlnet.gov/scorm/) content packaging or IMS Common Cartridge (http://www.imsglobal.org/cc/) that are typically supported by existing authoring tools could help in transferring a course structure with references to smart content to a number of different delivery platforms.

5.5 Supporting a Community of Practice

In addition to the technical issues that often obstruct adoption of SLCs, there are also non-technical obstacles to overcome. Providing the necessary community support to foster adoption should be taken seriously by all developers of SLCs. Some of the early activities typically carried out by SLC developers to foster a community of users include offering introductory workshops or tutorials to introduce others to their SLC; and providing online resources such as web sites, forums, or mailing lists to help new users connect with each other. While these steps are certainly necessary, fostering an active community often requires significantly greater investment. In this section, we list additional non-technical strategies that SLC developers could use to create and grow a more cohesive community of practice centered around their SLC.

Adopt a marketing plan.

Marketing is essential for developing and maintaining a community of practice. Developers of SLCs often know little about marketing and have even less affinity for its practices. So, they do not pay much attention to marketing their SLC. Marketing does not have to mean spending money to hire a marketing specialist, or even developing a full-blown marketing plan. It could be as simple as developing a thumbnail marketing plan, implementing it and continually revisiting it as SLC project matures.

Act as a technology evangelist.

A *technology evangelist* takes personal responsibility for continually promoting an SLC, its effectiveness, and its user base. Identifying one or more evangelists can be a critical element of a project's marketing plan. Some of the best evangelists are current users of the SLC who can speak with experience and authority about the usefulness and effectiveness of the SLC.

Plan for longevity.

Many academic research projects have short lifespans because they are primarily tied to short-term research funding. The authors are perfectly willing to share their work, but may offer little or no support, little or no active promotion or marketing, and no new features or enhancements once the short-term funding is exhausted. Since dissemination and adoption of an SLC takes time, cessation of support may come at about the same time an SLC project begins to see adoption and active use. This will preempt or sig-

nificantly dampen adoption of the SLC. To build a strong user community, it is necessary for a project team to adopt a long-term plan for continued development, ongoing support, and engagement of evangelists.

Streamline installation and setup.

Many academic research projects focus mainly on internal or local use of an SLC, primarily for the purposes of research evaluation and paper publication, with somewhat less of a focus on minimizing adoption barriers at other institutions and providing adopter support. One area where this becomes apparent is in the installation or setup process for an SLC: the more streamlined and less burdensome the process (e.g., no unnecessary steps, external dependencies, or prerequisite knowledge needed), the greater the chances of adoption and use.

Establish a brand identity.

While branding is an essential part of any commercial marketing plan, it is not considered seriously by many academic researchers. However, developing a brand identity is a powerful marketing tool—it makes the SLC more recognizable; makes it more easily searchable on the web; and makes all the dissemination, training, and community development efforts instantly recognizable. Branding can start with the most obvious steps: a project logo, a dedicated domain name, and a tag line for use on the web site and in product literature. How much further to take branding is a matter of choice and available resources, but could include conference sponsorships, brand-emblazoned giveaways, etc.

Tie in with mainstream course resources, like a specific textbook.

Developing synergistic relationships with mainstream learning resources (e.g., textbooks, classroom technologies) would provide significant dissemination and adoption leverage. It could ease adoption through piggybacking, and gain following by adding value to widely used mainstream learning resources.

By using some or all of these strategies, developers of SLCs can position themselves to be much more successful at dissemination than purely academic research projects. These strategies will lead to wider dissemination and use, and help build a community of practice around an SLC. A larger and stronger user community will in turn contribute to ongoing improvements of the SLC and make its dissemination self-sustaining.

6. SUMMARY AND DISCUSSION

In this paper, we have analyzed the field of smart learning content (SLC) for computing education. We set out to investigate the challenges associated with using, authoring, and developing SLCs, as well as related software support. We planned to discuss problems that inhibit adopting, sharing and customizing such content, and propose new technical solutions to overcome the challenges.

Our investigation led us to propose a definition of smart learning content (SLC) as being on a continuum along three axes: input, processing, and output. We used this definition in subsequent sections to categorize instances of SLCs that we identified from our literature survey and from educators through an online survey.

Next, based on our online survey of computer science educators, we summarized the types of SLCs used most frequently by instructors, problems educators have with adopting and using SLCs, and how educators typically integrate SLCs into their courses. We also assembled a list of SLCs mentioned by the respondents and then augmented it through our literature review to identify recently used SLCs. The collected SLCs were classified along the three axes of our definition.

In addition, we proposed a classification of architectures for hosting smart content based on the level of coupling between SLCs and their delivery platforms. Further, we proposed a categorization of the data collected by SLCs, according to the level of their granularity, and identified the purposes for which data at each level are used by various actors.

Next, we identified obstacles to integration and interoperability of SLCs, listed the advantages of integrating SLCs for various actors including teachers and students, and discussed the issues of conceptual integration, and problems with technological integration of delivery. This resulted in a proposal for an architecture that supports flexible reuse of SLCs, flexible course authoring, and flexible data collection, all of which are driven by considerations of loose coupling and reuse. We also addressed possible approaches to implement content discovery and personalization in this architecture. We proposed an ideal vision for embedding SLCs and storing data collected by them using new standards and protocols, followed by a pragmatic vision for achieving these goals based on existing, albeit imperfect, standards and protocols. Both visions are geared towards increasing the availability and adoption of SLCs. We also addressed the steps that can be taken to build a community of practice around an SLC.

6.1 Limitations

Our report has several limitations. To start with, we should acknowledge a relatively limited scope of the SLC review that served as the basis for categorizing pedagogical and technical aspects of SLCs. The scope of the review was defined by the timeframe of our working group. A more thorough review of the field would be a helpful addition as part of future work. At the same time, we believe that using a combination of the most popular and the most recent SLC systems helped to distill the most essential features and problems of modern SLCs in the field of computing. We also hope that several categorizations of SLCs proposed in the report will help in the future attempts to analyze this field.

More broadly, it also is important to note that our report has been deliberately focused on technical issues preventing wider use of SLCs. However, these are not the only barriers. Socio-cultural issues also play an important role in propagating SLC use and a review of these issues also would be valuable. Or report only examines a small number of these socio-cultural issues (such as community of practice). A broader set of these issues has been examined in prior work that is reviewed in the next subsection. However, a full survey remains as future work.

6.2 Comparison with Prior Work

Many researchers have been working on different aspects of SLC creation and adoption. Many surveys covering spe-

cific areas of SLCs, such as algorithm visualization [78], program visualization [81], and automatic program assessment systems [1, 38], as well as programming support tools and environments [50, 67] have been carried out. Such surveys have focused on specific issues of the denoted SLC subareas. While some previous ITICSE working groups [72, 55] have listed a large number of SLCs, we are not aware of any comprehensive survey of SLCs, and we recognize that such a survey would be very wide. Our online survey augments previous surveys by giving some overview of SLCs currently used by educators and our literature survey covers some recent developments in the field. A comprehensive survey, however, would be outside the scope of our working group.

In contrast with previous surveys, we have analyzed SLCs in a novel way by building a new categorization approach that captures the smartness of the interaction process between the user and the tool. Previously, some taxonomies and categorization schemes have been presented in SLC subareas, such as [50, 69], but prior approaches focus on specific issues in the subarea each addresses, whereas we take a more general view. On the other hand, Ihantola et al. [41], while discussing algorithm visualization, do take a more general view. They identify several dimensions in their taxonomy of effortlessness, including scope (a tool can be lesson-specific, course-specific, domain-specific and no-specific) and integrability (e.g., installation, customization, platform independence, internationalization, course management support and integration to hypertext). Their third dimension is interaction, though from a somewhat different point of view to ours, where they discuss the producer-system interaction and visualization-consumer interaction.

Several previous ITiCSE working groups discussing issues in smart content [72, 73, 55] have carried out online surveys for educators on various issues of using and adopting SLCs. Our work here augments these previous survey results. The previous working group reports have also proposed some technical architectures and/or guidelines for improving interoperability of SLCs. Roessling, Malmi, et al. [72] provided only guidelines to be considered in developing SLCs, while Roessling, Crescenzi, et al. [73] discussed issues in integrating SLCs with Moodle. Korhonen, Naps, et al. [55] focused on interactive computer science books (icseBooks) and presented an architecture for such books and discussed in some detail what support is needed for different stakeholders. They also discussed customization, peer review, versioning, authentication and authorization, as well as data collection in the context of icseBooks. While our work certainly overlaps with their work, there are clear differences as well. We are not focusing on building from scratch a single entity (book) with smart learning content, but we take a stand on how existing content could be made more accessible, more easily integrated into other learning resources, and more customizable in a distributed setting. Moreover, the previous work did not consider the aspects of user modeling, personalization, and adaptive learning content, although we aimed specifically to include these into our work. We provide a more advanced architecture and a much more comprehensive and technical analysis of the problem at hand.

6.3 Future Work

One of the most significant problems with adoption and reuse of SLCs today seems to be lack of communication: developers build SLCs mostly for their own use. They do not often invest in discoverability, integration, or interoperability of their SLC, reducing its chances of being adopted by other educators. Educators have a hard time finding SLCs that they can use in their courses. Even when they do find SLCs, they may be reluctant to use them for fear of significant commitment of time and effort to use the SLC.

At the same time, researchers have been building prototypes of delivery platforms, user models, and other components meant to facilitate discovery and integration of SLCs. However, they have often been using these as proof-of-concept systems, failing to graduate them into production systems, and failing to reach out to developers of SLCs who might benefit by integrating their SLCs with these systems. This situation is not expected to improve unless there is a meeting of the minds—of developers, educators and researchers. This working group report is a step in bringing together these three constituencies to start a dialog.

One future activity aimed at bridging this communication gap might be to hold a week-long hands-on workshop of developers, researchers, and educators where developers implement standard protocols so that their SLCs work with the services developed by researchers; and educators verify the utility of the SLC and its integration with such services.

In addition, this working group report lays out a clear architectural goal to strive toward in order to facilitate SLC integration in modern course management tools and learning platforms. The vision for this architecture was presented in Section 5. Initially, short-term progress can be made by developing a simple LTI-compliant adapter that can serve as a go-between allowing SLCs to be embedded in platforms that support LTI, as described in Section 5.4. This can be combined with efforts to enhance existing SLC projects to use a common learning record store backend for improved data analytics accessibility. These short-term steps can significantly increase opportunities for integrating SLCs into modern course delivery platforms with noticeably reduced effort. Longer term, educational researchers striving to improve SLC access can work toward the more idealized version of the architecture presented in Section 5.1. The notion of an SLC discovery tool, or even an SLC brokering service, can also be developed around the ideas presented in Section 5.2. While achieving the entire vision presented here will take significant effort over a long period of time, these steps also provide for near-term benefits for both SLC developers and CS educators who wish to employ smart learning content in their courses.

7. ACKNOWLEDGMENTS

The authors appreciate and acknowledge the feedback provided by David Hovemeyer, York College of Pennsylvania, Jaime Spacco, Knox College, IL, Cliff Shaffer, Virginia Tech, and Benjamin Valdes, Tecnológico de Monterrey.

This work is supported in part by the National Science Foundation under grants DUE-1245589 and DUE-0817187. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

8. REFERENCES

[1] K. M. Ala-Mutka. A Survey of Automated Assessment Approaches for Programming Assignments. *Computer Science Education*, 15(2):83–102, 2005.

[2] E. Allen, R. Cartwright, and B. Stoler. Drjava: A lightweight pedagogic environment for java. In *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '02, pages 137–141, New York, NY, USA, 2002. ACM.

[3] F. J. Almeida Martínez, J. U. Fuentes, and Ángel Velázquez Iturbide. VAST: Visualization of Abstract Syntax Trees Within Language Processors Courses. In *Proceedings of the 4th ACM Symposium on Software Visualization*, SoftVis '08, pages 209–210, New York, NY, USA, 2008. ACM.

[4] M. Ben-Ari, R. Bednarik, R. Ben-Bassat Levy, G. Ebel, A. Moreno, N. Myller, and E. Sutinen. A decade of research and development on program animation: The Jeliot experience. *Journal of Visual Languages & Computing*, 22(5):375–384, 2011.

[5] B. S. Bloom. *Taxonomy of educational objectives: The classification of educational goals. Handbook I: Cognitive domain*. New York: David McKay Company, 1956.

[6] C. Bonwell and J. Eison. *Active Learning: Creating Excitement in the Classroom AEHE-ERIC Higher Education Report No. 1*. Jossey-Bass, Washington, D.C., 1991.

[7] C. Brown, R. Pastel, B. Siever, and J. Earnest. JUG: A JUnit Generation, Time Complexity Analysis and Reporting Tool to Streamline Grading. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '12, pages 99–104, New York, NY, USA, 2012. ACM.

[8] P. Brusilovsky. KnowledgeTree: A Distributed Architecture for Adaptive e-Learning. In *Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers &Amp; Posters*, WWW Alt. '04, pages 104–113, New York, NY, USA, 2004. ACM.

[9] P. Brusilovsky. *Adaptive Hypermedia for Education and Training*, pages 46–68. Cambridge University Press, 2012.

[10] P. Brusilovsky, D. Baishya, R. Hosseini, J. Guerra, and M. Liang. KnowledgeZoom for Java: A Concept-Based Exam Study Tool with a Zoomable Open Student Model. In *Advanced Learning Technologies (ICALT), 2013 IEEE 13th International Conference on*, pages 275–279. IEEE, July 2013.

[11] P. Brusilovsky and T. D. Loboda. WADEIn II: A Case for Adaptive Explanatory Visualization. *SIGCSE Bull.*, 38(3):48–52, June 2006.

[12] P. Brusilovsky, E. Schwarz, and G. Weber. ELM-ART: An intelligent tutoring system on world wide web. In C. Frasson, G. Gauthier, and A. Lesgold, editors, *Intelligent Tutoring Systems*, volume 1086 of *Lecture Notes in Computer Science*, pages 261–269. Springer Berlin Heidelberg, 1996.

[13] P. Brusilovsky, S. Sosnovsky, and O. Shcherbinina. User modeling in a distributed e-learning architecture. In *User Modeling 2005*, pages 387–391. Springer, 2005.

[14] P. Brusilovsky, S. Sosnovsky, M. V. Yudelson, D. H. Lee, V. Zadorozhny, and X. Zhou. Learning SQL Programming with Interactive Tools: From Integration to Personalization. *Trans. Comput. Educ.*, 9(4):1–15, Jan. 2010.

[15] S. Bryfczynski, R. P. Pargas, M. M. Cooper, M. Klymkowsky, and B. C. Dean. Teaching Data Structures with beSocratic. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '13, pages 105–110, New York, NY, USA, 2013. ACM.

[16] L. M. S. Cámara, M. P. Velasco, and J. Ángel Velázquez Iturbide. Evaluation of a Collaborative Instructional Framework for Programming Learning. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '12, pages 162–167, New York, NY, USA, 2012. ACM.

[17] D. R. Cerezo, M. G. Albarrán, and J. L. S. Rodríguez. Interactive Educational Simulations for Promoting the Comprehension of Basic Compiler Construction Concepts. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '13, pages 28–33, New York, NY, USA, 2013. ACM.

[18] C. Cooper. *Individual Differences*. Oxford Illustrated Press, Oxford, UK, 1997.

[19] J. H. Cross, T. D. Hendrix, L. A. Barowski, and Others. Combining Dynamic Program Viewing and Testing in Early Computing Courses. In *Computer Software and Applications Conference (COMPSAC), 2011 IEEE 35th Annual*, pages 184–192. IEEE, 2011.

[20] J. H. Cross, T. D. Hendrix, J. Jain, and L. A. Barowski. Dynamic Object Viewers for Data Structures. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '07, pages 4–8, New York, NY, USA, 2007. ACM.

[21] P. Dillenbourg. *Collaborative Learning: Cognitive and Computational Approaches*. Advances in Learning and Instruction Series. Elsevier Science, Inc., New York, NY, 1999.

[22] S. H. Edwards. Improving student performance by evaluating how well students test their own programs. *J. Educ. Resour. Comput.*, 3(3), Sept. 2003.

[23] S. H. Edwards. Using software testing to move students from trial-and-error to reflection-in-action. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '04, pages 26–30, New York, NY, USA, 2004. ACM.

[24] S. H. Edwards and M. A. Pérez-Quiñones. Experiences using test-driven development with an automated grader. *J. Comput. Sci. Coll.*, 22(3):44–50, Jan. 2007.

[25] S. Esper, S. R. Foster, and W. G. Griswold. CodeSpells: Embodying the Metaphor of Wizardry for Programming. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '13, pages 249–254, New York, NY, USA, 2013. ACM.

[26] R. Farzan and P. Brusilovsky. Social navigation support in a course recommendation system. In *Adaptive hypermedia and adaptive web-based systems*, pages 91–100. Springer, 2006.

[27] R. M. Felder. Learning and Teaching Styles in Engineering Education. *Engineering Education*, 78(7):674–681, 1988.

[28] E. Fouh, V. Karavirta, D. A. Breakiron, S. Hamouda, S. Hall, T. Naps, and C. A. Shaffer. Design and architecture of an interactive eTextbook—The OpenDSA system. *Science of Computer Programming*, 88:22–40, Aug. 2014.

[29] M. Goldweber, R. Davoli, and T. Jonjic. Supporting Operating Systems Projects Using the ÂţMPS2 Hardware Simulator. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '12, pages 63–68, New York, NY, USA, 2012. ACM.

[30] J. Hattie and H. Timperlay. The Power of Feedback. *Review of Educational Research*, 77:81–112, 2007.

[31] J. Helminen, P. Ihantola, V. Karavirta, and L. Malmi. How do students solve parsons programming problems?: An analysis of interaction traces. In *Proceedings of the Ninth Annual International Conference on International Computing Education Research*, ICER '12, pages 119–126, New York, NY, USA, 2012. ACM.

[32] C. Higgins, T. Hegazy, P. Symeonidis, and A. Tsintsifas. The coursemarker cba system: Improvements over ceilidh. *Education and Information Technologies*, 8(3):287–304, 2003.

[33] M. Hilton and D. S. Janzen. On Teaching Arrays with Test-driven Learning in WebIDE. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '12, pages 93–98, New York, NY, USA, 2012. ACM.

[34] D. Hovemeyer, M. Hertz, P. Denny, J. Spacco, A. Papancea, J. Stamper, and K. Rivers. CloudCoder: Building a Community for Creating, Assigning, Evaluating and Sharing Programming Exercises (Abstract Only). In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, SIGCSE '13, page 742, New York, NY, USA, 2013. ACM.

[35] I.-H. Hsiao, F. Bakalov, P. Brusilovsky, and B. König-Ries. Progressor: social navigation support through open social student modeling. *New Review of Hypermedia and Multimedia*, 19(2):112–131, June 2013.

[36] I.-H. Hsiao, S. Sosnovsky, and P. Brusilovsky. Guiding students to the right questions: adaptive navigation support in an E-Learning system for Java programming. *Journal of Computer Assisted Learning*, 26(4):270–283, 2010.

[37] J. Hyvönen and L. Malmi. TRAKLA - A System for Teaching Algorithms Using Email and a Graphical Editor. In *Proceedings of HYPERMEDIA in Vaasa*, pages 141–147, 1993.

[38] P. Ihantola, T. Ahoniemi, V. Karavirta, and O. Seppälä. Review of Recent Systems for Automatic Assessment of Programming Assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, Koli Calling '10, pages 86–93, New York, NY, USA, 2010. ACM.

[39] P. Ihantola, J. Helminen, and V. Karavirta. How to Study Programming on Mobile Touch Devices: Interactive Python Code Exercises. In *Proceedings of the 13th Koli Calling International Conference on Computing Education Research*, Koli Calling '13, pages 51–58, New York, NY, USA, 2013. ACM.

[40] P. Ihantola and V. Karavirta. Two-Dimensional Parson's Puzzles: The Concept, Tools, and First Observations. *Journal of Information Technology Education: Innovations in Practice*, 10:1–14, 2011.

[41] P. Ihantola, V. Karavirta, A. Korhonen, and J. Nikander. Taxonomy of effortless creation of algorithm visualizations. In *Proceedings of the First International Workshop on Computing Education Research*, ICER '05, pages 123–133, New York, NY, USA, 2005. ACM.

[42] S. A. Jalil, B. Plimmer, I. Warren, and A. L. Reilly. Design Eye: An Interactive Learning Environment Based on the Solo Taxonomy. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '13, pages 22–27, New York, NY, USA, 2013. ACM.

[43] P. Jarusek and R. Pelánek. A Web-based Problem Solving Tool for Introductory Computer Science. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '12, page 371, New York, NY, USA, 2012. ACM.

[44] J. Jeuring, A. Gerdes, and B. Heeren. Ask-Elle: A Haskell Tutor. In A. Ravenscroft, S. Lindstaedt, C. Kloos, and D. Hernández-Leo, editors, *21st Century Learning for 21st Century Skills*, volume 7563 of *Lecture Notes in Computer Science*, pages 453–458. Springer Berlin Heidelberg, 2012.

[45] M. Joy, N. Griffiths, and R. Boyatt. The boss online submission and assessment system. *Journal on Educational Resources in Computing (JERIC)*, 5(3):2, 2005.

[46] V. Karavirta, P. Ihantola, and T. Koskinen. Service-Oriented Approach to Improve Interoperability of E-Learning Systems. In *Advanced Learning Technologies (ICALT), 2013 IEEE 13th International Conference on*, ICALT '13, pages 341–345, Washington, DC, USA, July 2013. IEEE.

[47] V. Karavirta, A. Korhonen, L. Malmi, and K. Stalnacke. MatrixPro - A Tool for Demonstrating Data Structures and Algorithms Ex Tempore. In *Proceedings of the IEEE International Conference on Advanced Learning Technologies*, ICALT '04, pages 892–893, Washington, DC, USA, 2004. IEEE Computer Society.

[48] V. Karavirta and C. A. Shaffer. JSAV: The JavaScript Algorithm Visualization Library. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '13, pages 159–164, New York, NY, USA, 2013. ACM.

[49] J. Kay, B. Kummerfeld, and P. Lauder. Personis: A Server for User Models. In P. De Bra, P. Brusilovsky, and R. Conejo, editors, *Adaptive Hypermedia and Adaptive Web-Based Systems*, volume 2347 of *Lecture Notes in Computer Science*, pages 203–212. Springer Berlin Heidelberg, 2002.

[50] C. Kelleher and R. Pausch. Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers. *ACM Comput. Surv.*, 37(2):83–137,

June 2005.

[51] Kiosked. Smart content trend report 2013, 2013. [Online; accessed 10-July-2014].

[52] K. R. Koedinger, R. Baker, K. Cunningham, A. Skogsholm, B. Leber, and J. Stamper. A data repository for the EDM community: The PSLC DataShop. *Handbook of educational data mining*, 43, 2010.

[53] M. Kölling, B. Quig, A. Patterson, and J. Rosenberg. The BlueJ System and its Pedagogy. *Computer Science Education*, 13(4):249–268, Dec. 2003.

[54] A. Korhonen, J. Helminen, V. Karavirta, and O. Seppälä. Trakla2. In A. Pears and C. Schulte, editors, *Proceedings of the 9th Koli Calling International Conference on Computing Education Research*, pages 43–46. University of Joensuu, Nov. 2010.

[55] A. Korhonen, T. Naps, C. Boisvert, P. Crescenzi, V. Karavirta, L. Mannila, B. Miller, B. Morrison, S. H. Rodger, R. Ross, and C. A. Shaffer. Requirements and Design Strategies for Open Source Interactive Computer Science eBooks. In *Proceedings of the ITiCSE Working Group Reports Conference on Innovation and Technology in Computer Science Education-working Group Reports*, ITiCSE -WGR '13, pages 53–72, New York, NY, USA, 2013. ACM.

[56] A. Kumar. A Scalable Solution for Adaptive Problem Sequencing and Its Evaluation. In V. Wade, H. Ashman, and B. Smyth, editors, *Adaptive Hypermedia and Adaptive Web-Based Systems*, volume 4018 of *Lecture Notes in Computer Science*, pages 161–171. Springer Berlin Heidelberg, 2006.

[57] A. N. Kumar. Results from the Evaluation of the Effectiveness of an Online Tutor on Expression Evaluation. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '05, pages 216–220, New York, NY, USA, 2005. ACM.

[58] A. N. Kumar. A Study of the Influence of Code-tracing Problems on Code-writing Skills. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '13, pages 183–188, New York, NY, USA, 2013. ACM.

[59] J. Kurhila, M. Miettinen, P. Nokelainen, and H. Tirri. Educo - A Collaborative Learning Environment Based on Social Navigation. In P. De Bra, P. Brusilovsky, and R. Conejo, editors, *Adaptive Hypermedia and Adaptive Web-Based Systems*, volume 2347 of *Lecture Notes in Computer Science*, pages 242–252. Springer Berlin Heidelberg, 2002.

[60] A. D. Learning. Trainging & learning architecture (TLA): Learning record store, 2014. [Online; accessed 10-July-2014].

[61] T. R. Liyanagunawardena, A. A. Adams, and S. A. Williams. MOOCs: A systematic study of the published literature 2008-2012. *The International Review of Research in Open and Distance Learning*, 14(3):202–227, 2013.

[62] T. MacWilliam and D. J. Malan. Streamlining Grading Toward Better Feedback. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '13, pages 147–152, New York, NY, USA, 2013. ACM.

[63] Y. Matsuzawa, K. Okada, and S. Sakai. Programming Process Visualizer: A Proposal of the Tool for Students to Observe Their Programming Process. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '13, pages 46–51, New York, NY, USA, 2013. ACM.

[64] T. L. Naps, J. R. Eagan, and L. L. Norton. JHAVÉ&Mdash;an Environment to Actively Engage Students in Web-based Algorithm Visualizations. *SIGCSE Bull.*, 32(1):109–113, Mar. 2000.

[65] T. L. Naps, J. R. Eagan, and L. L. Norton. JhavéâĂŤan environment to actively engage students in web-based algorithm visualizations. *ACM SIGCSE Bulletin*, 32(1):109–113, 2000.

[66] M. C. Orsega, B. T. Vander Zanden, and C. H. Skinner. Two experiments using learning rate to evaluate an experimenter developed tool for splay trees. In *Proceedings of the 42Nd ACM Technical Symposium on Computer Science Education*, SIGCSE '11, pages 135–140, New York, NY, USA, 2011. ACM.

[67] A. Pears, S. Seidman, L. Malmi, L. Mannila, E. Adams, J. Bennedsen, M. Devlin, and J. Paterson. A Survey of Literature on the Teaching of Introductory Programming. In *Working Group Reports on ITiCSE on Innovation and Technology in Computer Science Education*, ITiCSE-WGR '07, pages 204–223, New York, NY, USA, 2007. ACM.

[68] M. C. Polson and J. J. Richardson. *Foundations of intelligent tutoring systems*. Psychology Press, 2013.

[69] B. A. Price, R. M. Baecker, and I. S. Small. A Principled Taxonomy of Software Visualization. *Journal of Visual Languages and Computing*, 4(3):211–266, 1993.

[70] M. Procopiuc, O. Procopiuc, and S. H. Rodger. Visualization and Interaction in the Computer Science Formal Languages Course with JFLAP. In *Frontiers in Education Conference*, 1996.

[71] T. Rajala, M. J. Laakso, E. Kaila, and T. Salakoski. VILLE: A Language-independent Program Visualization Tool. In *Proceedings of the Seventh Baltic Sea Conference on Computing Education Research - Volume 88*, Koli Calling '07, pages 151–159, Darlinghurst, Australia, Australia, 2007. Australian Computer Society, Inc.

[72] G. Rössling, M. Joy, A. Moreno, A. Radenski, L. Malmi, A. Kerren, T. Naps, R. J. Ross, M. Clancy, A. Korhonen, R. Oechsle, and Ángel Velázquez Iturbide. Enhancing Learning Management Systems to Better Support Computer Science Education. *SIGCSE Bull.*, 40(4):142–166, Nov. 2008.

[73] G. Rössling, M. McNally, P. Crescenzi, A. Radenski, P. Ihantola, and M. G. Sánchez-Torrubia. Adapting Moodle to Better Support CS Education. In *Proceedings of the 2010 ITiCSE Working Group Reports*, ITiCSE-WGR '10, pages 15–27, New York, NY, USA, 2010. ACM.

[74] G. Rössling, M. Schüer, and B. Freisleben. The ANIMAL Algorithm Animation Tool. *SIGCSE Bull.*, 32(3):37–40, July 2000.

[75] A. L. Santos. An Open-ended Environment for Teaching Java in Context. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '12, pages 87–92, New York, NY, USA, 2012. ACM.

[76] M. Scaife and Y. Rogers. External cognition : how do graphical representations work? *Int . J . Human âĂŞ Computer Studies*, 45:185–213, 1996.

[77] Schafer, D. Frankowski, J. Herlocker, and S. Sen. Collaborative Filtering Recommender Systems. In P. Brusilovsky, A. Kobsa, and W. Nejdl, editors, *The Adaptive Web*, volume 4321 of *Lecture Notes in Computer Science*, chapter 9, pages 291–324. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[78] C. A. Shaffer, M. L. Cooper, A. J. Alon, M. Akbar, M. Stewart, S. Ponce, and S. H. Edwards. Algorithm Visualization: The State of the Field. *Trans. Comput. Educ.*, 10(3), Aug. 2010.

[79] T. Sirkiä. A JavaScript Library for Visualizing Program Execution. In *Proceedings of the 13th Koli Calling International Conference on Computing Education Research*, Koli Calling '13, pages 189–190, New York, NY, USA, 2013. ACM.

[80] D. Skrien. CPU Sim 3.1: A Tool for Simulating Computer Architectures for Computer Organization Classes. *J. Educ. Resour. Comput.*, 1(4):46–59, Dec. 2001.

[81] J. Sorva, V. Karavirta, and L. Malmi. A Review of Generic Program Visualization Systems for Introductory Programming Education. *Trans. Comput. Educ.*, 13(4), Nov. 2013.

[82] J. Sorva and T. Sirkiä. UUhistle: A Software Tool for Visual Program Simulation. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, Koli Calling '10, pages 49–54, New York, NY, USA, 2010. ACM.

[83] S. Sosnovsky, P. Brusilovsky, M. Yudelson, A. Mitrovic, M. Mathews, and A. N. Kumar. Semantic Integration of Adaptive Educational Systems. In T. Kuflik, S. Berkovsky, F. Carmagnola, D. Heckmann, and A. Krüger, editors, *Advances in Ubiquitous User Modelling*, volume 5830 of *Lecture Notes in Computer Science*, chapter Semantic Integration of Adaptive Educational Systems, pages 134–158. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[84] S. Sosnovsky and T. Gavrilova. Development of Educational Ontology for C-programming. *International Journal on Information Theories & Applications*, 13(4):303–308, 2006.

[85] J. Spacco, D. Fossati, J. Stamper, and K. Rivers. Towards Improving Programming Habits to Create Better Computer Science Course Outcomes. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '13, pages 243–248, New York, NY, USA, 2013. ACM.

[86] J. Sweller. Cognitive Load Theory, learning difficulty, and instructional design. *Learning and Instruction*, 4(4):295–312, 1994.

[87] J. Urquiza-Fuentes and J. A. Velázquez-Iturbide. A Survey of Successful Evaluations of Program Visualization and Algorithm Animation Systems. *Trans. Comput. Educ.*, 9(2), June 2009.

[88] K. Veeramachaneni, Z. Pardos, U.-M. O'Reilly, F. Dernoncourt, and C. Taylor. MOOCdb: Developing Data Standards and Systems for MOOC Data Science. In *1st Workshop on Massive Open Online Courses at the 16th Annual Conference on Artificial Intelligence in Education, AIED 2013*, 2013.

[89] J. A. Velázquez Iturbide. Refinement of an Experimental Approach Tocomputer-based, Active Learning of Greedy Algorithms. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education*, ITiCSE '12, pages 46–51, New York, NY, USA, 2012. ACM.

[90] A. Vihavainen, T. Vikberg, M. Luukkainen, and M. Pärtel. Scaffolding Students' Learning Using Test My Code. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '13, pages 117–122, New York, NY, USA, 2013. ACM.

[91] L. S. Vygotsky. *The history of the development of higher mental functions*, volume 4 of *The collected works of L. S. Vygotsky*. Plenum Press., New York, 1997.

[92] M. Yudelson, P. Brusilovsky, and V. Zadorozhny. A User Modeling Server for Contemporary Adaptive Hypermedia: An Evaluation of the Push Approach to Evidence Propagation. In C. Conati, K. McCoy, and G. Paliouras, editors, *User Modeling 2007*, volume 4511 of *Lecture Notes in Computer Science*, chapter 6, pages 27–36. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2007.

[93] B. V. Zanden, D. Anderson, C. Taylor, W. Davis, and M. W. Berry. Codeassessor: An Interactive, Web-based Tool for Introductory Programming. *J. Comput. Sci. Coll.*, 28(2):73–80, Dec. 2012.

APPENDIX

A. EXAMPLES OF CS SMART LEARNING CONTENT

In this appendix, we provide descriptions of selected resources in computer science that may be appropriately called smart learning content (SLC). Many of these examples reflect the work of members of this ITiCSE working group, and are presented as examples only to illustrate the variety present among SLCs, instead of being intended as a comprehensive list of all SLC work.

A.1 Algorithm and Program Visualization

A.1.1 JSAV and OpenDSA

JSAV is a JavaScript implementation of algorithm visualization exercises used in TRAKLA2 [48]. JSAV supports creation of of algorithm visualizations with different levels of user interaction, automated assessment of students solutions, embedding such smart content to online material and collection of interaction data. JSAV runs also on mobile touch devices and approximately half of the students prefers solving algorithm visualization exercises on mobile devices [48, 55]. JSAV is used also in the OpenDSA (`http://algoviz.org/OpenDSA/`) open source project that seeks to provide complete instructional materials for data structures and algorithms (DSA) courses [28]. OpenDSA eTextbooks can be tailored with the content selected by an instructor, and OpenDSA has been used at all levels of the computer science curriculum. OpenDSA eTextbooks integrate textbook-quality content with algorithm visualizations for all algorithms and many automatically assessed interactive exercises. These include standard multiple choice-style questions, small programming exercises, and algorithm proficiency exercises. In algorithm proficiency exercises, students are shown a data structure in a graphical interface, and must manipulate it to demonstrate knowledge of an algorithmic process. For example, they might show the swap operations that a given sorting algorithm uses.

Figure 10: OpenDSA aims to make it easy for an instructor to create a custom interactive textbook for his or her class.

OpenDSA uses a "content server" to deliver content to a student's browser in HTM5, with score data and user interaction log data sent to a "scoring server" for grading and storage. Most OpenDSA visualization and exercises support the OEmbed protocol, and so can be integrated easily within other web content. OpenDSA supports customization with extensive internationalization support, and dynamic switching of example programming language.

A.1.2 JSVEE

JSVEE (JavaScript Visual Execution Environment) [79] library provides an easy way to publish interactive program execution visualizations (see Figure 11). By using the library, HTML5-based animations can be embedded in web pages together with the text and other learning materials. Animations help students to understand how the program is executed by showing a graphical representation of the computer's memory and executing small code snippets by animating all the steps of the execution. Animations show, for example, how parameter passing, stack, heap, objects, references and other important concepts work which are normally completely invisible to the students but still very important concepts to understand.

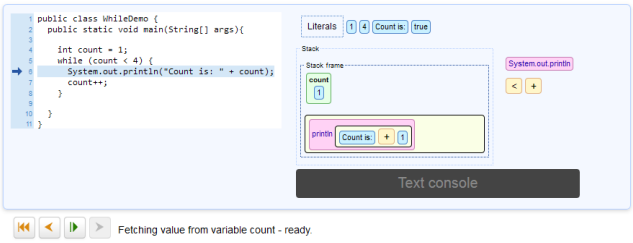

Figure 11: An interactive animation of a simple while loop made with the JSVEE library.

A.2 Automatic assessment tools

A.2.1 Web-CAT

Web-CAT (`http://web-cat.org/`) is a web-based automated grading system for programming assignments [24, 23, 22]. It uses a plug-in-based architecture to provide for customizability and extensibility. While the server itself is implemented in Java, student assignments are processed by plug-ins, and new programming languages can be supported by creating new plug-ins. Many of Web-CAT's benefits stem from allowing students to experience multiple feedback/revision cycles before completing their assignments, so that they may incorporate the results from automated feedback into their solution development process rather than only receiving feedback after the assignment is finished. While instructors can use reference tests to grade student work, Web-CAT is most well-known for allowing instructors to grade students based on how well they test their own code. Web-CAT is distributed for free as an open-source project.

A.2.2 Test My Code

Test My Code [90] is a service that is used to both assess students' work and to scaffold students as they are working on programming assignments. The service contains backend services that are used to provide scoreboard details, create courses, create embeddable questionnaires, monitor students' progress, create code reviews, and store snapshot

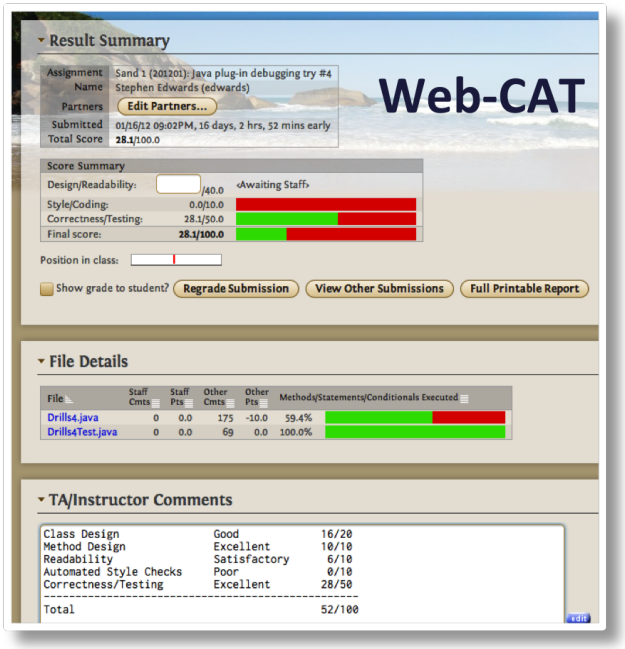

Figure 12: Web-CAT

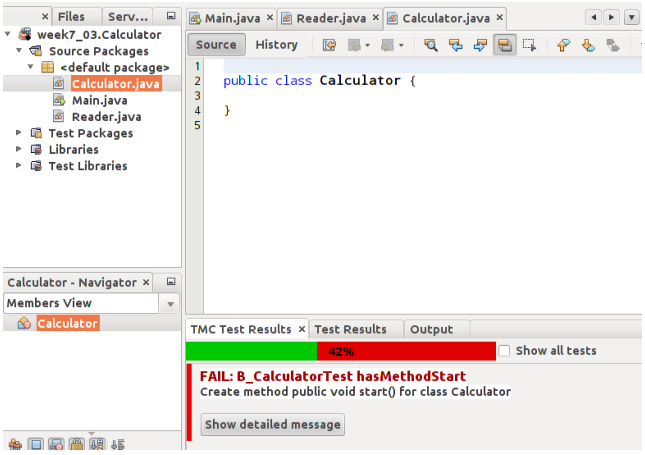

Figure 13: Test My Code

data from the students progress. Students working with Test My Code typically utilize an IDE plugin that provides authentication, functionality for downloading and submitting programming assignments, functionality for gathering feedback from the assignments (feedback questions created within the backend), functionality for providing hints or scaffolding messages that can be used to redirect students' work, and asking and receiving code reviews as well as key-level logging. Test My Code also provides a web-based editor that can be embedded to a web page.

A.3 Coding Tools

A.3.1 Codecademy

Codecademy (`http://www.codecademy.com`) is a web portal containing interactive learning materials for programming in many languages, such as Python, Ruby, PHP, Javascript and HTML/CSS. Tutorials are completed by following step-by-step instructions and writing the required code in the code editor. The code is executed inside the environment and therefore it is an easy way to learn programming languages in the browser without installing any external software.

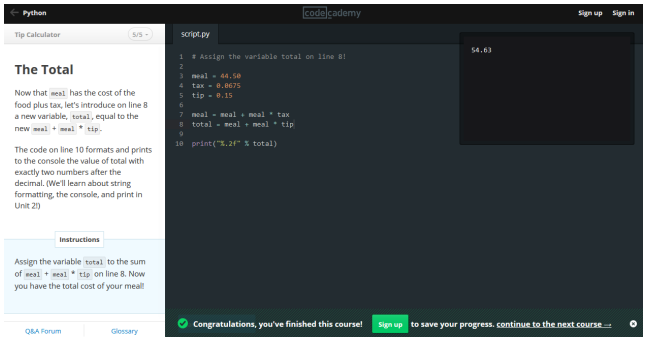

Figure 14: An interactive coding session in Codecademy. The instructions are on the left side and coding area on the right side. The output of the program is also visible to check the results.

A.4 Algorithm and program simulation tools

A.4.1 UUhistle

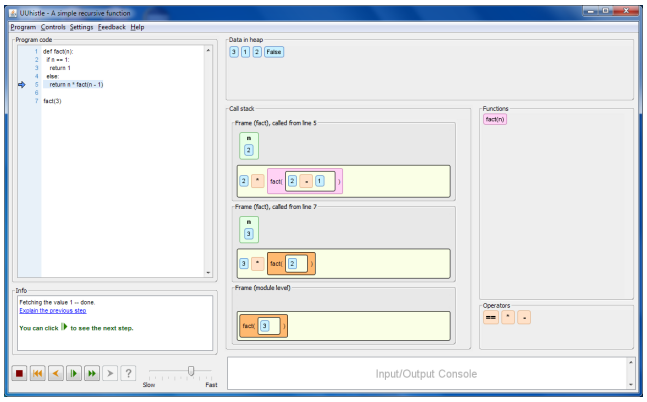

Figure 15: UUhistle is executing a recursive function call.

UUhistle[82] is a program visualization and simulation application for novice programmers. It can visualize the execution of the given Python programs but in addition to this, it can also be used to create visual program simulation exercises in which students take the role of the computer and execute the program by dragging and dropping the visual elements. In this way students are enforced to think

constantly the correct execution model instead of passively watching the animation. UUhistle can be embedded as Java applet to web pages or used as an independent application.

A.5 Problem-solving support tools

A.5.1 js-parsons

Js-parsons (Figure 16), is used to give automated feedback from visual code construction exercises [40]. These are a type of scaffolded program construction tasks where the learner is given a set of code fragments, blocks of a single or multiple lines of code, and from these the task is to piece together a program. In js-parsons environment, learners not only select and order, but also indent and edit code fragments. In Python, code indentation has a semantic meaning. That is, code blocks are defined by indentation instead of start and end symbols such as curly braces. We have found this kind of exercises well suited to mobile gadgets where the challenge is to effectively create and adapt content to touch devices and overcome the restrictions of these novel learning platforms [39]. Detailed logs can be used to analyze students' problem solving processes [31]. The system is used at Aalto and at the University of North Carolina at Charlotte, as well as in the interactive version of *How to Think Like a Computer Scientist: Learning with Python* (`http://interactivepython.org/runestone/static/thinkcspy/index.html`).

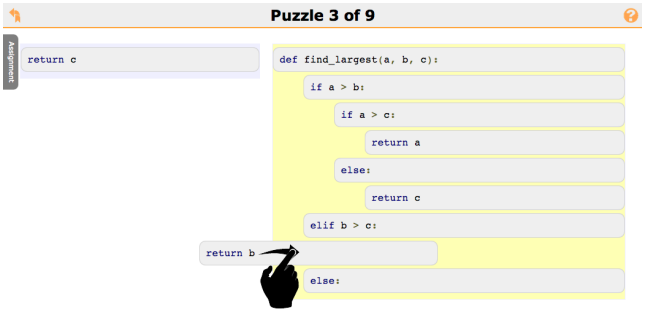

Figure 16: Code construction in js-parsons mobile environment.

A.5.2 Problets

Problets (`http://problets.org`) are software tutors designed to help students learn programming concepts by solving problems. They are meant to be used as supplements to classroom instruction and complements to traditional programming projects. The types of problems presented by problets include expression evaluation, debugging, code tracing and specifying program state. Problets are available for all the topics typically covered in introductory programming courses in C++, Java and C#. Problets provide step-by-step explanation of the correct solution to each problem, which has been shown to improve learning. Problets adapt to the learner, minimizing the number of problems solved while maximizing learning. They have been used over the web by educators at dozens of institutions continually for ten years as of Spring 2014.

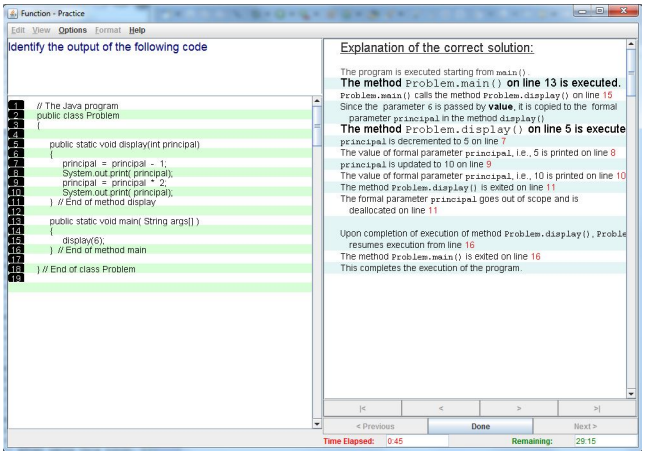

Figure 17: Snapshot of Code Tracing Problet in Java: Problem shown in the left panel and step-by-step explanation shown in the right panel.

B. SURVEY RESULTS

Table 2 lists the systems reported in the online survey (labeled as O) as well as the systems extracted from last two years of ITiCSE (labeled as L). The systems have been first classified using the dimensions of smart learning content described in Section 2, after which pedagogical foundations (discussed in Section 3.3) have been added to the systems. Although the dimensions are not binary per se, the categorization is presented in a binary fashion. Here, for example, if a system has some free-form text inputs, it will be considered as a system that processes free-form input.

The labels for the dimensions are based on the categories *Input* (Pre-specified (P) / Free-form (F)), *Process* (Not computational (Nc) / Fully computational (Fc), and *Output* (Generic (G) / Customized (C)).

Shorthands for the pedagogical basis are as follows: *Collaborative learning* (CI), *Active learning* (Al), *Individual differences* (Id), *External representations* (Er), and *Feedback* (Fb).

System / SLC and Reported Usage	Survey	Dimensions			Pedagogical foundations				
	L / O	P / F	Nc / Fc	G / C	CL	AL	ID	ER	Fb
Animal [65] (AV)	O	F	Fc	C	0	1	0	0	1
Ask-Elle [44] (AA)	O	P	Fc	G	0	1	0	1	0
Auto marked python exercises (AA)	O	F	Fc	C	0	1	0	0	0
BEAST (Sim)	O	F	Fc	C	0	1	0	0	1
BlueJ [53] (PV)	O	F	Fc	C	0	1	0	1	1
Cloudcoder [34] (Coding)	O	F	Fc	C	0	1	0	1	0
CodeAssessor [93] (AA,Coding)	O	F	Nc	C	0	1	0	1	0
Codecademy [`http://www.codecademy.com/`] (Other)	O	F	Fc	C	0	1	0	1	0
CodeHS [`http://codehs.com/`] (Coding)	O	F	Fc	C	0	1	0	0	1
CPU Sim [80] (Sim)	O	F	Fc	C	0	1	0	0	1
DrJava [2] (PV)	O	F	Fc	C	0	1	0	0	1
DrRacket [`http://racket-lang.org/`] (PV)	O	F	Fc	C	0	1	0	0	1
Edline [`https://www.edline.net/`] (AA)	O	P	Fc	G	0	1	0	0	0
GreedEx [89] (AV,Sim)	O	F	Fc	C	0	0	0	0	1
Insertion sort animation at Wikipedia (AV)	O	P	Fc	G	0	1	0	1	1
CodingBat [`http://codingbat.com`] (PV,AA,Coding,Problem,Others)	O	F	Fc	C	0	1	0	0	1
Jeliot [4] (AV,PV)	O	F	Fc	C	1	1	0	1	0
JFLAP [70] (AV,PV,Sim)	O	F	Fc	C	0	1	0	1	0
jFlex (Sim)	O	F	Fc	C	0	0	0	1	0
jGRASP [20] (PV)	O	F	Fc	C	0	1	0	0	1
jHave [64] (AV)	O	F	Fc	C	0	1	1	0	1
KhanAcademy [`https://www.khanacademy.org/`] (Coding)	O	F	Fc	C	0	1	0	0	1
MatrixPro [47] (AV)	O	F	Nc	C	0	1	1	1	0
Mooshak [`https://mooshak.dcc.fc.up.pt/`] (AA)	O	F	Fc	C	0	1	0	0	1
MyProgrammingLab [`http://www.pearsonmylabandmastering.com/northamerica/myprogramminglab/`] (Coding)	O	F	Fc	C	1	0	0	0	0
OpenDSA [28] (AV,Sim)	O	F	Fc	C	0	1	0	1	0
Practice-It! [`http://practiceit.cs.washington.edu/`] (Other)	O	F	Nc	C	0	1	0	1	0
Problets [57] (Problem)	O	P	Fc	C	0	0	0	1	0
Programming Course Resource System (Coding)	O	F	Fc	C	0	1	0	0	1
Pythontutor.com [`http://pythontutor.com/`] (PV)	O	F	Fc	C	0	1	0	0	1
Sketchmate [66] (AV,Sim)	O	F	Fc	C	0	1	0	1	0
Sorting Animation applets (AV)	O	P	Fc	G	0	1	0	1	0
Sphero (AV,PV)	O	F	Fc	C	0	1	0	1	0
Testing locally developed scripts (AA)	O	F	Fc	C	0	1	0	0	0
testpilot (AA)	O	P	Fc	G	0	1	0	1	0
TRAKLA [37] (AV)	O	F	Fc	C	0	1	0	1	0
TRAKLA2 [54] (AA,Sim)	O	F	Fc	C	0	1	0	1	0
Turingscraft CodeLab [`http://turingscraft.com/`] (Coding)	O	F	Fc	C	0	1	0	1	0
UUhistle [82] (Sim)	O	F	Fc	C	0	1	0	0	1
VAST [3] (Sim)	O	F	Fc	C	0	0	0	1	0
ViLLE [71] (PV,AA,Coding,Sim,Other)	O	F	Fc	C	0	1	0	0	1
WadeIn [11] (Problem)	O	F	Fc	C	0	1	0	0	1
Web-CAT [24, 23, 22] (AA)	O	F	Fc	C	0	1	1	0	1
AGUIA/J [75] (Coding)	L	F	Fc	C	0	1	0	1	0
beSocratic GraphPad [15] (Other)	L	F	Fc	C	0	1	0	1	1
CodeSpells [25] (Coding)	L	F	Fc	C	0	1	0	1	1
CS50 Submit [62] (AA)	L	F	Nc	C	0	0	0	0	0
Design Eye [42] (Other)	L	P	Nc	G	1	1	0	1	1
Evaluators 2.0 [17] (Sim)	L	P	Fc	C	0	1	0	1	1
JSAV [48] (PV)	L	P	Fc	C	0	1	1	1	1
JUG [7] (AA)	L	F	Fc	C	0	1	1	1	1
Marmoset [85] (AA)	L	F	Fc	C	0	1	0	0	1
MoCAS [16] (Other)	L	F	Nc	C	1	1	0	0	1
Problem Solving Tutor [43] (AA)	L	P	Fc	C	0	1	1	1	1
Programming Process Visualizer [63] (PV)	L	F	Fc	C	0	1	1	1	0
Test My Code [90] (AA)	L	F	Fc	C	0	1	0	0	1
WebIDE [33] (Coding)	L	F	Fc	C	0	1	0	1	1
μMPS2 hardware simulator [29] (Sim)	L	P	Fc	C	0	1	0	0	0

Table 2: Systems extracted both from the online survey (O) and the literature survey (L) identifying recent advances in smart learning content. The systems are categorized both based on their dimensions (see Section 2) and their pedagogical basis (see Section 3.3). Participants of the online survey were able to report systems under the following categories: Algorithm Visualization (AV), Program Visualization (PV), Automatic Assessment (AA), Coding, Simulation (Sim), and Other. The categories reported in surveys appear in parentheses after each system name. It should be noted that these categories are only those reported by respondents, which may not reflect all the features of the SLCs. For SLCs identified only from the literature, the reported usage category is based on how the system is described in the article.

C. SURVEY INSTRUMENT: SMART CONTENT IN CS EDUCATION

This survey is being conducted by the ITICSE 2014 Working Group on "Increasing Accessibility and Adoption of Smart Technologies for Computer Science Education". With the survey, we hope to gather information about the following:

- What kinds of Smart Content (SC) are being used by computing educators?
- What issues do teachers and students face in adopting and using SCs?
- What needs and visions do teachers and students have when using SCs?

This information will help the Working Group draft its report, which may be included in ACM InRoads magazine.

For the purposes of this survey, we define smart content (SC) as follows: Smart content is active educational content "beyond simple files". It has at least two of the following features:

- SC works with students interactively
- SC provides students feedback based on (some) of their actions
- SC collects data about students' interactions with it
- SC remembers its users and allows them to resume from where they left off rather than start from scratch
- SC accumulates and displays students' progress
- SC can adapt to the students' interests and level of knowledge

Note that smart content might be delivered in different modes: e.g., as downloadable application, web service, or cloud service.

We invite you to fill out this survey. Your participation in the survey from the perspective of a Computer Science educator, whether or not you have used smart content in the recent past, would help us get a more complete picture of the use of smart content in Computer Science education. We are also interested in your observations of your students' use of smart content.

The survey contains two (short) parts:

1. Using smart content
2. Visions and challenges

We expect that the survey will take up to 10 minutes to fill out.

We would appreciate your participation by Friday, June 20th, 2014. However, if you miss this date, we would still be interested in your participation, since we plan to continue to collect data beyond that date.

Part 1: Using smart content

What CS-specific SCs have you used since 2010? Select for each category that applies.

	never	have TRIED myself but never used in my course	have SUGGESTED as optional learning resource for my students	have REQUIRED students to use in my course
Algorithm visualizations (e.g., jHave, Animal)				
Program visualizations (e.g., BlueJ, jGRASP)				
Automatic assessment tools (e.g., CourseMarker, Boss)				
Coding tools (e.g., CodeLab, CloudCoder)				
Simulation tools (e.g., TRAKLA2, UUhistle)				
Problem-solving software (e.g., problets, WadeIn)				
Something else (specify later)				

Names of SCs you have SUGGESTED or REQUIRED

If you have SUGGESTED or REQUIRED SCs in your course since 2010, please, list their names here.

- Algorithm visualization SC you/your institution developed: (e.g., jHave, Animal)
- Algorithm visualization SC others developed: (e.g., jHave, Animal)
- Program visualization SC you/your institution developed: (e.g., BlueJ, jGRASP)
- Program visualization SC others developed: (e.g., BlueJ, jGRASP)
- Automatic assessment SC you/your institution developed: (e.g., CourseMarker, Boss)
- Automatic assessment SC others developed: (e.g., CourseMarker, Boss)
- Coding SC you/your institution developed: (e.g., CodeLab, CloudCoder)
- Coding SC others developed: (e.g., CodeLab, CloudCoder)
- Simulation SC you/your institution developed: (e.g., TRAKLA2, UUhistle)
- Simulation SC others developed: (e.g., TRAKLA2, UUhistle)
- Problem-solving SC you/your institution developed: (e.g., problets, WadeIn)
- Problem-solving SC others developed: (e.g., problets, WadeIn)
- SC of a category not listed above, you/your institution developed:
- SC of a category not listed above, others developed:

Part 2: Visions and challenges

Describe what kinds of SCs you would like to use in your courses, in terms of covered content area and functionality.

Describe how you would like to integrate SCs into your courses from a pedagogical point of view, such as for open labs, self study, classroom teaching or closed labs. Would they be for assignments, demonstrations, tests etc.? If you use more than one category of SC (e.g., program visualization, algorithm visualization, etc.), please respond to each category separately.

What challenges and difficulties have you faced in adopting SCs? Please, select each which applies.

- Difficulty finding SCs which I could use
- Difficulty getting the SCs to work in my own / my students' environments
- I cannot customize the SCs to fit my local needs
- I cannot integrate the SCs into my other learning resources
- I cannot integrate the SCs with other systems in my institution, e.g., to store results in grade roster
- Organizing student authentication has been laborious or problematic
- I am concerned about security of the collected student data
- My students need to use too many different SCs The SCs are not aligned with my way of teaching
- Other:

Have you used data collected by SCs? If yes, what data and for what purposes you have used it?