An Introductory Course on Service-Oriented Computing for High Schools

W. T. Tsai, Yinong Chen, Calvin Cheng, and Xin Sun
School of Computing and Informatics, Arizona State University, Tempe, AZ, USA

Wtsai@asu.edu; Yinong.Chen@asu.edu; Calvin.Cheng@bannerhealth.com; Xin.Sun@asu.edu

Gary Bitter and Mary White
College of Education, Arizona State University, Tempe, AZ, USA

Bitter@asu.edu; Mary.White@asu.edu

[The contents of this paper were developed under the Fund for the Improvement of Postsecondary Education (FIPSE), U.S. Department of Education. However, these contents do not necessarily represent the policy of the Department, and you should not assume endorsement by the Federal Government.]

Executive Summary

Service-Oriented Computing (SOC) is a new computing paradigm that has been adopted by major computer companies as well as government agencies such as the Department of Defense for mission-critical applications. SOC is being used for developing Web and electronic business applications, as well as robotics, gaming, and scientific applications. Yet, SOC education is lagging. In spite of significant progresses in SOC technology and applications, SOC education has not been taught in introductory classes, even if it is technologically feasible. Most of the existing SOC courses are either graduate seminars or senior-level courses at various universities. On the other hand, SOC is component-based and introduces a high-level of abstraction, which makes it possible to teach computing within an application domain, such as robotics. In other words, the SOC learning process focuses more on the application logic rather than the syntax of programming languages, potentially making computer education entertaining. The Computer Science & Engineering Department and the College of Education at Arizona State University, in cooperation with the Scottsdale Unified School District and Coronado High School, pioneered the first SOC course for high schools (grades 9 through 12) in the Spring and Fall of 2007. The course was also offered in summers 2006, 2007, and 2008 to high school students and teachers. This course is designed so that even high school students without any computing training can learn the latest software technologies in an entertaining manner. Specifically, this course uses a visual programming environment so students can design their software visually. Thus, they can learn control constructs but do not need to focus on the details of programming language syntax. Furthermore, once the software is designed, a visual

Editor: Janice Whatley

3-D simulator can give students instantaneous feedback. Finally, the software designed can be downloaded into LEGO robots for robotic competition. Most students who took the course rated the course "very well" or "pretty well" and some indicated that they are interested enough to contemplate pursuing computer science as their career.

Keywords: Service Oriented Computing, Service Oriented Architecture, High School Computer Classes, Teacher Education, Computer Science, Information Technology Education, Curricular Reform

Introduction

Computer science education in high schools can be traced back to the 1980s when the procedural programming language Pascal was identified as the language of choice. In the 1990s, when the object-oriented computing paradigm dominated the computing industry, many high schools started teaching the object-oriented programming languages C++ and then Java.

A computing paradigm refers to the set of concepts, principles, and methods of expressing computation that allows human command on one hand and allows the computer to efficiently execute on the other hand. The major computing paradigms that have dominated computer science education and software development are:

- **Imperative computing** developed in the 1950s. FORTRAN was the flagship language of this approach, which dominated software development in the 50s and 60s and extended even into 70s.
- **Procedural computing** developed during the 1970s and 80s, which emphasized modular software design. Programming languages Algol, Pascal, and C represent this approach.
- **Object-Oriented Computing (OOC)** developed after the 80s, with an emphasis on abstraction and code reuse. Representative object-oriented programming languages include C++, Java, and C#.
- **Service-Oriented Computing (SOC),** also known as Service-Oriented Architecture (SOA), started in late 1990s. Java and C# are the two major programming languages on which SOC is built. However, SOC uses existing services to compose applications. The key to enable pervasive code reuse is the standardization of service interfaces leading to platform-independent code reuse. Incorporated with the Internet and search engines, service discovery is vital to SOC (Chen & Tsai, 2008; Eposito, 2004; Heinemann & Rau, 2003; Singh & Huhns, 2005).

SOC has been described as a method to build distributed applications more effectively. According to Singh and Huhns (2005), "Many of the key techniques now being applied in building services and service-based applications were developed in the areas of databases, distributed computing, artificial intelligence, and multi-agent systems. These are generally established bodies of work that can be readily adapted for service composition." Their book on SOC describes, in part, how services can be used to facilitate simpler kinds of composition. SOC allows the user to engage a service so that the relevant computational themes are "peer-to-peer computing, messaging, transactions (traditional, nested, and extended or relaxed), workflow, [and] business processes." These computational themes allow for ready-transfer into a high school or university classroom.

Phelps and Busby (2007) define SOC as providing "well-defined business functions as services, which are made available to multiple applications through standard protocols." They claim that SOC is a fundamental shift in the way applications are built. This shift requires a new way of thinking about processes and the subsequent role that applications play.

SOC became important around 2000-2001 when all the major computer companies, including BEA, HP, IBM, Intel, Microsoft, Oracle, and SAP agreed on a set of standard interfaces, which make it possible for software developed in any language and on any platform to communicate and cooperate with each other. Furthermore, SOC is component-based and introduces a high-level of abstraction. It has the potential to become the first computing course that shifts the course content from the syntax of programming languages to the problem solving process.

In spite of significant progress in SOC applications and its potential being taught as the first computing course, SOC education is currently lagging. Johnson and Liber (2008) discuss a model called Personal Learning Environment (PLE) "as a practical intervention concerning the organization of technology in education." The model relies on SOC to address the issue of engagement with tools "by allowing learners to control their own instrumentation." They also consider SOC a paradigm shift that moves the locus of control to the learner. They state that "the ways in which learners exercise that control becomes an important educational issue" and include issues of self-efficacy.

Many universities have started to offer SOC courses. However, the SOC is often not taught in introductory computing courses in spite of the fact that SOC concepts of using existing services to compose applications are simpler, more intuitive, and more exciting than traditional computing, thus SOC can be taught in introductory courses. The authors have pioneered the first SOC class for high schools. Sponsored by the U.S. Department of Education, the authors taught the course to 30 high-school students in grades 9 through 12 and in the Spring and Fall of 2007 (SRLAB, 2008b). Three summer programs were also offered in 2006, 2007, and 2008 (SRLAB, 2008c). Sixty-one high-school students and twenty five teachers participated in the programs. A number of high schools in Arizona have started to offer this course as their regular course since Spring 2008 (Arizona State University, n.d.). This paper discusses the design of the course and reports the preliminary evaluation of the class.

The rest of the paper is organized as follows: The next section addresses the importance and motivation of SOC; the third section highlights the differences between traditional computing education and the new SOC education and discusses various approaches to make the SOC high school education feasible. The fourth through seventh sections present the proposed course contents, present the course projects used in the course, present the course evaluation, and state the major lessons learned.

Service-Oriented Computing

SOC refers to the set of concepts, principles, and methods that represent computing in a collection of loosely coupled services. The distinguishing feature of software applications in SOC is that they are constructed from component services with standard interfaces (Chen & Tsai, 2008) that make such components reusable.

Previously, software was developed using high-level programming languages such as C++ and Java. Software reuse is limited in spite of the significant progress in reusability technologies such as data abstraction, separation of implementation from its specification, object-oriented classes, design patterns, code templates, and object-oriented frameworks. SOC can be considered an extension of these software reusability concepts. SOC emphasizes specification of services using open standards, discovery of services in repositories using specification only, creation of composite services using existing services, and dynamic application composition, runtime management by policies. While some of these concepts were known previously, SOC packages them together and requires a new mindset to develop software:

- In the early days, software developers thought in terms of control constructs and subroutines (procedural paradigm);
- Later they thought in terms of classes, instances, methods and dynamic binding (object-oriented paradigm);
- In SOC, they need to think in terms of services, workflows, and dynamic management and control (SOC paradigm).

Each new computing approach never replaces an existing approach as they can complement each other for an extended period of time, but new engineers need to have a new mindset to develop software in a different way.

SOC emphasizes software development by composition and reuse. In SOC, software development involves three parties: application builders, service providers, and service brokers. Service providers use a programming language such as Java or C# to write program components. All components are wrapped with open standard interfaces to form services or Web services, so that application builders can use those services across the Internet. The same services can be utilized by many application builders. Service brokers allow services to be registered and published for public access, so application builders can find the services they need. Application builders are software engineers who have a good understanding of software architecture and of the application domain (i.e. end users' needs). Figure 1 shows the three parties involved with common SOC protocols.

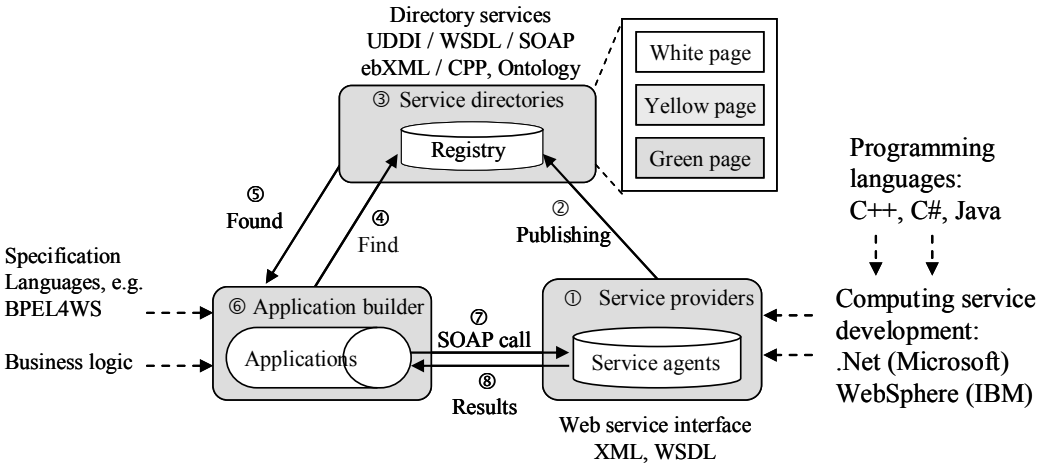


Figure 1. Three Parties of Service-Oriented Computing and their Common Protocols

A unique feature of SOC is that application builders no longer construct software from scratch using a programming language. Instead, they specify the application logic in a high-level workflow language, utilizing standard services as components. Service providers are like manufacturers of LEGO pieces, whereas application builders are like users of LEGO pieces who build a large number of different LEGO-based applications/products for different uses. In the future, the market will need more people who can use services to compose and use applications, rather than those who develop specific software services. This is similar to the case where there are more people who use LEGO pieces than people who develop and manufacture those pieces.

SOC is often strongly connected to applications such as e-business, robotics, healthcare, retail, and telecommunication, and this makes learning SOC concrete, relevant, challenging, and exciting.

Growing Demand of the SOC Workforce

SOC requires the service interface design to follow open standards so every service developed can be shared in a service repository possibly connected over the Internet. The accumulation of available services over time will make it possible to find almost any desired component services. Thus, the job market demand is shifting towards high-level composition modeling skills for application building through composition of reusable services. This has a huge implication not well known today – potentially, not only will computer majors need to learn SOC, even non-majors may need to learn SOC as they will use SOC services in daily applications. This is similar to the development and manufacturing of automobiles—the general public learns to drive automobiles without necessarily knowing the technical details. Similarly, computer program developers need to develop SOC infrastructure and services for applications, while the general public may use these services without necessarily knowing the technical details of the SOC infrastructure.

Computer applications are penetrating numerous domains, and the need for application builders is increasing. We are witnessing "the largest labor force migration in history" (IBM Research, 2006), from manufacturing to services, which now account for more than 50 percent of the labor force in Brazil, Germany, Japan, and Russia, and for 75 percent of the labor force in the US and the UK. At the same time, all major computer corporations (e.g. IBM, Microsoft, Oracle, HP, SAP, Intel, Cisco, Juniper, Sun Microsystems, and BEA) have moved into SOC. For example, the ASP.Net framework is Microsoft's flagship for SOC development (Esposito, 2004; Nagel, 2005) on which multiple modeling languages and tools are being developed, including XLANG and Indigo/Windows Communication Foundation (Resnick, Crane, & Bowen, 2008). To facilitate SOC growth, Microsoft also started an eScience program to apply SOC in scientific areas such as physics, chemistry, astronomy, mathematics, geology, and biology (Microsoft, n.d.). IBM has restructured itself as a SOC company. WebSphere is IBM's flagship for SOC application development, and SCA/SDO (Service Component Architecture / Service Data Object) are its modeling languages. Other major corporations also created their SOC development frameworks, e.g., SAP Web Application Server (SAP, n.d.), SOA Suite 10g from Oracle, and Sun Java System Active Server Pages from Sun Microsystems (Sun Microsystems, n.d.). All these frameworks support BPEL (Business Logic Execution Language for Web Services), jointly developed by BEA Systems, IBM, Microsoft, Oracle, SAP AG, Siebel Systems (Wikipedia, n.d. a).

Furthermore, government agencies such as the U.S. Department of Defense (DoD), have adopted SOC. In fact, all of DoD's major IT initiatives in the last six years were based on SOC, including the Army's FCS, the Navy's FORCEnet, the Air Force's JBI, and the OSD's NCES and GIG-ES (Paul, 2005).

Additionally, major computer users such as American Express (finance), US Bank (banking), Office Depot (retail), and Amazon (online retailer) are also transitioning to SOC. Many Internet-based electronic business applications, including online banking, online shopping, travel agencies, and electronic commerce, have currently implemented SOC.

The Urgent Need for SOC Education

Most universities today continue to teach traditional programming courses, largely based on object-oriented computing, i.e. they cover programming language constructs and apply these programming constructs to develop objects and use objects to build programs for problem solving. This approach has served computing education well since the inception of computing and almost all introductory books on computing begin in this manner. However, this approach is not completely consistent with the SOC approach where software development is mainly composed using reusable services rather than writing code using a programming language.

Currently, most SOC courses at universities are in the experimental stage or have a seminar format. Although there is a clear and growing demand for SOC expertise, only a tiny fraction of students graduating with a degree in computer science and engineering in U.S. have the necessary SOC skills today.

Another serious problem currently experienced by most computer science and engineering programs in the U.S. is the significant decline in interest and enrollment. This crisis is partly due to the exporting of labor-intensive programming jobs and the layoff of local programmers. Prospective students and their parents are not informed about the current transition to SOC in the industries and the associated change in the job market, with the fast-growing demand for SOC-skilled workforce. An article entitled "Education Key to SOA Success" (LaPlante, 2005) reported that the biggest roadblock in the implementation of SOC, identified by 56 percent of the respondents to a recent survey, was the need of SOC education. For example, IBM recently proposed the SSME (Service Sciences, Management, and Engineering) program where SOC plays a key role (IBM Research, 2006).

SOC has the potential to attract people to learn more about computers and make computer technology their career. Paul Horn, a senior vice president in charge of the IBM labs pointed out that the IBM employees who were initially dubious about SOC gradually "got excited by working on the fascinating problems in services" (Lohr, 2006). What accounts for the attitude change among experts is that SOC involves "high-end work" that "typically taps several disciplines, requires conceptual thinking and pattern recognition" (Lohr, 2006) and is conducted to solve emerging problems in diverse application environments. These features of SOC maintain a high level of intellectual challenge, require creativity and constant learning, and make the SOC jobs exciting and intrinsically rewarding (in addition to the extrinsic motivation of high pay in a hot job market). Moving from individual benefits brought by SOC skills to national benefits, according to MIT economist Frank Levy (IBM Research, 2006), massive SOC education would help the U.S. economy to retain the most creative computing work that is hard to automate and outsource, adds the most value, and pays the highest wages.

IT educators need to be concerned with teaching SOC as it is often considered one of the latest innovations in IT today. They have ten outstanding reasons to do so (Gillard, Bailey, & Nolan, 2008) including their need to maintain leadership and set the example for their students. In fact, if they do not incorporate SOC into their curriculum it is likely their students will face professional challenges in the future.

Even though OOC has been the dominating computing approach since early 1980s, innovative ways of educating object-oriented concepts and programming is an active research area (Fjuk, Karahasanovic, & Kaasboll, 2006) today as educators discover new approaches to teach object-oriented concepts. If SOC education follows the path of OOC education, there will be a good amount of future research on SOC education.

Issues and Changes for SOC education

New Mindset for SOC

Often, initial computing courses have a significant impact on students. For example, in Spring 2005, the authors taught a SOC course at Arizona State University to thirty-five senior/graduate students (Tsai, Chen, et al., 2008). In the first project, only two out of the fifteen groups turned in an SOC design, and the rest turned in a traditional object-oriented design, proving that prior training and experience have oriented students to a certain design approach, and it was difficult for them to learn something new and different.

Similar phenomena have been encountered before during the previous paradigm shift. For example, when FORTRAN programmers were trained to construct structured programs using Pascal, they may have submitted a FORTRAN-like program using the Pascal syntax instead. This bias has been observed repeatedly for 40 years, and this generated numerous debates over five generations of programming languages including FORTRAN, Algol, PL/1, Pascal, C, Ada, Prolog, Smalltalk, C++, and Java.

Thus, the first computing framework taught often has a significant impact on the way people program (Jadud, 2003). Note that previously the debates mainly focused on programming styles (such as functional abstraction vs. data abstraction) or specific programming constructs (pointers vs. reference), yet SOC brings a new dimension, i.e. composition vs. construction, and this composition concept has not been adequately covered in today's computer science or computing courses. As a paradigm shift is eminent, we decided to teach the SOC course at the high school level. The advantages of this include helping students familiarize themselves with SOC before being affected by any previous computing paradigms, developing their interests for computer sciences, and preparing them for further study in colleges (Tsai, Chen, & Sun, 2007).

Teaching Strategies Shift for SOC

The differences between SOC and traditional programming are shown in Table 1, and the unique features of SOC education are shown in Table 2.

Table 1. Differences between Traditional Computing and Service-Oriented Computing

Strategic Difference	Traditional Programming Education	New SOC Education
Educational Goal	Learn programming language constructs and apply them for problem solving.	Learn the overall application architecture and how to compose applications using existing component services.
Focus	On hardware and software interface, system interaction, low-level programming techniques, and low-level reusability (rather than applications).	On service specification, application composition, human/computer interactions, system interaction, and software modules, applications domains, and high-level reusability.
Curriculum Content	The syntax of the programming language, with an emphasis on the construction of program modules.	The SOC principles and the use of existing services to compose applications.
Order of Curriculum Contents	Learn programming language constructs, followed by architecture design.	Learn software architecture design followed by workflows and services.
First Computing Course	Develop applications from scratch.	Develop applications by composition using existing services in an SOC infrastructure.

Table 2 Unique Features of SOC Education

Features	Traditional Programming	Service-Oriented Computing
Overall Process	For example, object-oriented design by first identifying data, classes, or associated methods.	Software development by identifying loosely coupled services and composing them into executable applications.
Level of Abstraction and Cooperation	Application development is often delegated to a single team responsible for the entire lifecycle of the application. Developers need to have programming knowledge and some domain knowledge.	Development is delegated to three independent parties: application builder, service provider, and service broker. Builders understand application logic and may not know how services are implemented. Providers develop services but may not know the applications.
Code Sharing and Reuse	Code reuse through inheritance of class members and through library functions. Often these are platform dependent.	Code reuse at service level. Services have standard interfaces and are published on Internet repository. They are platform-independent and can be searched and remotely accessed. Service brokerage enables systematic sharing of services.
Dynamic Binding and Re-composition	Associating a name to a method at runtime. The method must have been linked to the executable code before the application is deployed.	Binding a service request to a service can be done at the design time or at runtime. The services can be discovered after the application has been deployed. This feature allows an application to be composed (and re-composed) at runtime.

Software Development Environment for Introductory SOC Education

An important issue in teaching SOC is the necessity of using a commercial-grade tool, as SOC requires significant infrastructure and software tools to be useful. However, another important issue is that the tools should be easy to use. In fact, as the course is designed for high-school students, it is even important for the tools to be interesting and fun. Unfortunately, many existing SOC tools and software development environments are suitable for working professionals or advanced users only. Thus, they may not be suitable for teaching introductory computing course. For example, Table 3 shows the major SOC tools:

Table 3 Major SOC Tools Comparison

	Description	Vendor	Comments
Websphere	It has many SOC features, such as service composition, discovery, and modeling.	IBM	This tool is IBM's flagship SOC tool. It requires significant SOC knowledge.
.NET	This supports many features of SOC including service creation, discovery, composition, deployment.	Microsoft	This is Microsoft's flagship product for SOC. It is based on Microsoft's Windows system, and uses C# as its main programming language.
HP SOA Center	This supports SOC features such as service composition, modeling, integration, management (HP, n.d.).	HP	This is HP's flagship product for SOC. It requires significant SOC knowledge.

Weblogic	This supports service deployment, composition, management, policy (BEA, n.d).	BEA	This is BEA's flagship product for SOC. It is a heavy-weight environment, suitable for professional use.
Oracle SOA Suite	This supports service creation, deployment, composition, orchestration (Oracle, 2007).	Oracle	This is Oracle's flagship product for SOC. It requires a high-end server rather than a regular desktop, so it is suitable for professional use.
Enterprise SOA	This supports service creation, deployment, composition (SAP, n.d.).	SAP	This is SAP's flagship product and it is integrated with many of SAP's existing products. This tool is sophisticated and often requires a user to be familiar with SAP products.

Service Composition Languages are important for developing SOA applications; however, not all of them are suitable for teaching SOA to students with minimum computing knowledge. BPEL (Wikipedia, n.d. a), PSML (Tsai, Cao, et al., 2007), C# (Hasan & Duran, 2006) and VPL (Microsoft MSDN, n.d.) are four service composition languages available currently. Table 4 compares them based on SOA features supported and whether they are suitable for high school students. The first four criteria – service creation, service composition, service specification, and service collaboration – are standard SOC techniques. The visualized flowchart supports easy understanding of workflows, and visualized simulation is important for high school students to understand the behavior of software developed. Multiple vendor support is important for tool sustainability (in case the failure of a vendor) and neutrality among vendors. The last row is the summary of evaluation for using the specific tool/language for high school students. Some languages/tools have outstanding features, but they may be suitable for computer graduate students and working professionals only because significant software design knowledge is needed before using them.

Table 4 Service Composition Langages Comparison

	BPEL	**PSML**	**C#**	**VPL**
Service Creation	Yes	No	Yes	Yes
Service Composition	Yes	Yes	Yes	Yes
Service Specification	Yes	Yes	No	No
Service Collaboration	Yes	Yes	No	No
Visualized Flowchart	No	Yes	No	Yes
Visualized Simulation	No	No	No	Yes
Multiple vendor support	Yes	No	No	No
Suitable for High School Students	No	No	No	Yes

As the course under discussion is designed for high school students, the following features are considered important:

- Support basic SOC features including service specification, discovery, composition, and collaboration;
- Friendly GUI for easy visualization including visualization of SOC workflows and services;

- Support dynamic simulation, including visualized simulation;
- Integration of interesting applications.

After investigation, the authors used a visual software development environment, Microsoft Robotic Studio, to compose SOC applications. This toolset has many interesting features:

- The tool directly supports SOC and includes service creation, deployment, and composition. However, because this tool was released in December 2006, many advanced SOC features have not been developed yet. This is still acceptable as those missing features are probably more suitable for subsequent courses.
- An important feature is that the tool has a friendly GUI for visualization instead of command-line programming editors or even a modern integrated programming environment such as VC++, VB, or Delphi. Another unique feature is that workflow, services, and architecture can be visualized.
- The tool supports dynamic simulation (in an animated mode) and even allows the simulation run to be visualized. The visualization allows students to see the execution flow explicitly.
- The tool is fully integrated with VPL (Visual Programming Language) (Microsoft MSDN, n.d.) and it is a graphical dataflow language that is easy to learn.

These features make this robotic studio an interesting education for high school students to learn software design and have fun during the process. Specifically, they can learn robotic software design, and then they play robots controlled by the SOC software designed by them.

In summary, this new software development environment is different from the common high school computing environment, and it is even different from modern computing environments commonly used at universities today.

Exciting and Entertaining Applications

With this new environment, it is possible to teach students to drag-and-drop visual components to model applications with workflows and architecture, discover services, simulate applications, deploy applications, and view program execution all in a single integrated visual environment. While many of these individual capabilities were available before, they were scattered among different platforms written for particular programming languages and did not talk to each other. Furthermore, the software development process is interesting and entertaining as software is mainly composed rather than programmed.

For example, Figure 2 shows the use of the Microsoft Visual Studio .Net framework to develop an application. The programming task is to develop an online bookstore (e-business) that takes orders from clients, processes the orders, and places charges by using the services of remote banks. If a charge is successful, the desired book will be ordered from the publisher and delivered to the client. It is difficult to program such a system from scratch using C++, Java, or even C#. However, if all the components (boxes in Figure 2) can be found in the service repository, the development task in SOC is simple. What an application builder would do is (1) draw the model (Figure 2); (2) provide a text file containing the data to be changed between the services, for example, the data behind the message “getOrder” must contain the client’s name, address, credit card information; (3) define the logic and flow (for example, the node “processOrder” defines the execution order: first place the charge, then wait for the result of the operation; if the charge is successful, accept the order, otherwise, reject it).

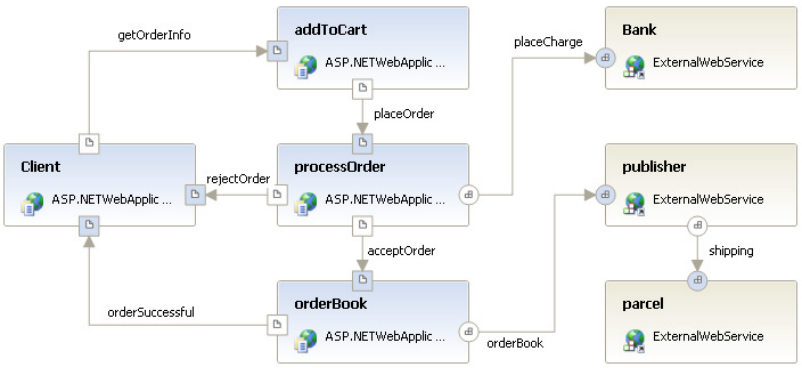


Figure 2. Visual Programs of the Online Bookstore

The use of entertaining applications to teach SOC is relevant here as well. Specifically, this project uses robotics, gaming, and e-business applications for teaching SOC to high-school students. Figure 3 shows the design diagram created using VPL (Visual Programming Language), which is based on drag-and-drop and SOC. On one hand, the graphic program has the same programming power as an ordinary program. On the other hand, the program can simply call services supplied by any parties including robotic vendors. In the program in Figure 3, an application is constructed by simply dragging and dropping services from the service directory provided in the framework. All the services in the service directory are either developed by Microsoft, robotics companies, or individual users and can be invoked both locally and remotely. An application like this required thousands of lines of code when SOC was not available. However, with the concept of SOC, even a high-school student having no previous programming experience can complete this application on his/her own.

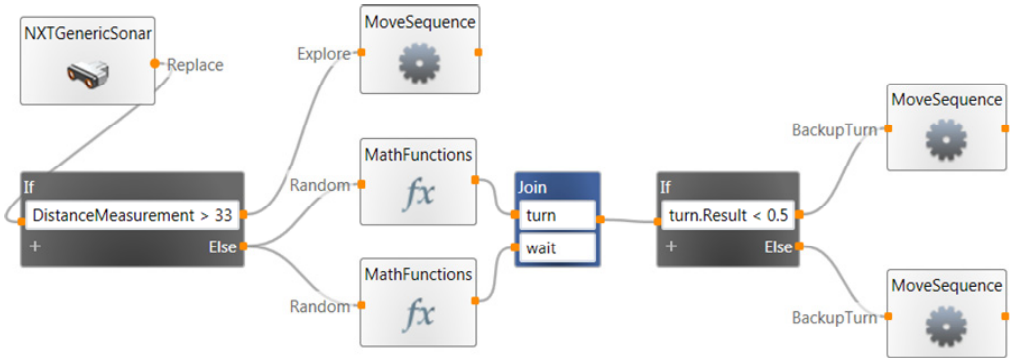


Figure 3. Visual Program for a Robot Control Application

Figure 3 shows the design for a robot control application, specifically it controls the movement of a robot in a maze. In the design diagram in Figure 3, a service is used to wrap a sonar sensor, which returns the distance measurement to the obstacle. Based on different reading values, the program takes different actions by calling different services. In this example, if the distance is greater than 33 cm, the robot continues to explore the maze (see Figure 4). Otherwise, it calls a mathematical function to decide whether to make a left-turn or a right turn. VPL is an executable visual programming language, which means the graphic design shown in Figure 3 is ready to compile and run. As can be seen in this example, the program does not require as much programming detail as ordinary programming languages. Students can focus on the logic in visual form and let the compiler translate the logic into the executable.

Microsoft Robotics Studio also provides a powerful simulation environment that realistically shows the application in a virtual space. With this simulation environment, the students can create physical entities, such as sky, ground, light, and various geometric shapes, and combine them all. With proper meshes and services, they can create life-like simulated robots with which they can run and test their programs. This turns the boring traditional programming into an easy, fun, game-like development process. The students can see the amazing results in the simulation environment immediately when they finish their program, and this gives them confidence and stimulates interest to complete more complicated applications.

This was well demonstrated by the authors' experimental classes. When we showed the students the program in the simulation environment, the students were excited. And because of this interest and the ease of the development tool, they can quickly learn the SOC concepts and apply them to construct robotic software. For example, students can program robots that can go around in a maze, as shown in Figure 4. Before SOC was introduced in robotics, these kinds of programming tasks could only be done by people with extensive computer hardware and software knowledge (such as embedded system programming). But now, high school students without prior computer training can do it within a semester. We have written a text book (Chen & Tsai, 2008) in which the concepts, tutorials, and hands-on laboratories in Alice and Robotics Studio are explained in detail.

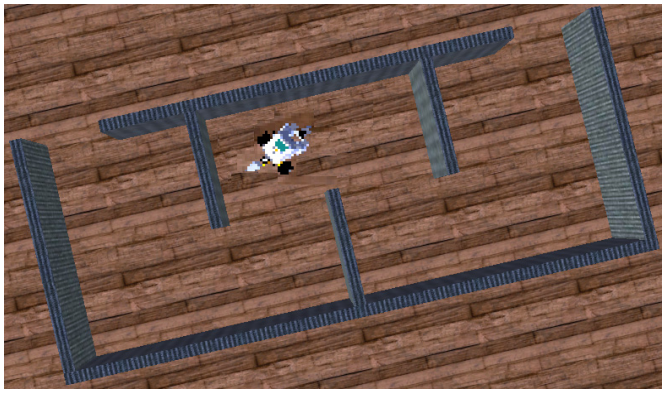


Figure 4. Simulation of Robot Finding its Way out of a Maze

One issue in SOC education is to keep the class vendor neutral. As SOC is still considered new, with few commercial platforms available, even fewer platforms are available for introductory courses where ease of use is a critical issue. While in previous experience we used Microsoft tools, this project used open source versions from Microsoft as much as possible. For example, instead of using the proprietary version, this project used the Shared Source Common Language Infrastructure (SSCLI) (Wikipedia, n.d., b) which is the open source version of the .NET Common Language Infrastructure.

Other SOC Education Issues

Software service availability

An important issue is the availability of sufficient services for SOC education. It is necessary to have a reasonable size of service repository for service discovery, navigation, and reuse. Actually, several repositories of reliable services are available, for example:

- Amazon Web Service Developer Connection at http://Developer.amazonwebservices.com

- XML Web Services Repository at www.xmlwebservices.cc
- UDDI service registry at www.uddi.org
- Xmethod service registry at xmethods.net/
- IBM UDDI at http://www-306.ibm.com/software/solutions/webservices/uddi/
- Services that come with the installation of Robotic Studio

Many of these services are open source, so their reliability can be easily determined. Many vendors also provide their own repositories of services. We also developed quite a few services using VPL for the students to use, which can be found at (SRLAB, 2008c). The students can use VPL to compose new services and applications in a simple drag-and-drop style using the default services and developed services.

Quality and reliability of services used

Another important issue is that services need to be reliable enough to be used in applications. In other words, services need to be verified and validated; otherwise application builders cannot use them. Numerous approaches are available at this time. While this is still an active research area, several approaches are currently available:

- The first approach is to allow open source services only as the source code is available, and people can perform independent verification and validation. However, this approach is expensive and will not allow use of those services published without source code for mission-critical applications. Also, to protect their own interests, many organizations may not be willing to publish their services with source code.
- The second solution is to allow services from business partners because business partners share source code verification. This approach is taken by some computer and software companies, and the quality of services essentially depends on their reputation and warranty.
- The third approach, proposed in *Service-Oriented System Engineering: A New Paradigm* (Tsai, 2005) is to have Collaborative Verification and Validation (CV&V) of services by all parties, including service clients, service providers, and service brokers. Services that pass verification and validation will have digital signatures to ensure that they cannot be changed after verification. This approach is different from traditional IV&V (Independent Verification and Validation) where independent parties perform verification tasks on software; in CV&V, all parties are encouraged to participate and contribute in a sharing environment to perform service verification.

Cross referencing to STEM knowledge

Another interesting issue is that once services are developed for an application, such as robotic games, these services can have service descriptions that reference common STEM (Science, Technology, Engineering, and Mathematics) knowledge (Tsai, Sun, et al., 2008). The services used may contain STEM content, and it is useful for students to learn or refresh relevant STEM knowledge while they discover services or compose applications. For example, in SumoBot, a popular robotic game, two robots compete to drive each other out of a ring, and students need to have a variety of STEM knowledge to discover services and compose applications. Specifically, students need to know the physics concepts (such as speed and acceleration), geometric concepts (such as circumference and radius of a circle), algebraic concepts (such as variables and functions), and game-theory concepts (such as anticipating opponent movement to develop a winning game). Thus, while students discover services related to SumoBot and compose an application

based on these services, they also learn or refresh relevant STEM knowledge. Table 5 shows common services useful for the SumoBot, and the relevant STEM knowledge, and Table 6 shows a sample service description that reference to a physics concept velocity.

Table 5. Common Robotic Service

Supporting Services	Description	Level & Knowledge
Distance Calculation	Calculates the distance based on the infrared sensor detection data.	Basic. Physics, Math.
Turning Angle Calculation	Calculates the proper turning angle when searching or chasing the opponents.	Basic. Math.
Turning Time Calculation	Calculates the time (ms) required to apply voltage on wheel control port to perform a turning action based on the turning mode and angle.	Medium. Physics, Mechanics, Math.
Voltage Calculation	Calculates the proper voltage required to keep the motor running at a certain speed during an active approaching phase.	Basic. Physics, Math.

Table 6. Sample Service Description

Service: This is a service that calculates the velocity of a robot.

Author: Xin Sun, Arizona State University, contributed this service in March 2008.

Goal, Usage and Ranking: This goal of this service is to estimate the velocity of a robot by giving the estimated distance and the estimated traveling time. Since this service has been available, it has been used by 150 unique robotic applications, 70 unique users, and over 120 times. This service is ranked number two among all the services that compute velocity and it is ranked 167th among all services.

Knowledge: Relevant knowledge of this service is Physics-mechanic and Mathematics-Algebra.

Mathematical formula used:

Speed or velocity = distance/travel time.

The shorter the distance, with the same travel time, the higher the speed. If the reader is interested in using an approach where a robot will travel at different speeds, they can use services such as Velocity2 and Velocity7. For detailed discussion on the knowledge, see the following websites at Wikipedia http://en.wikipedia.org/wiki/Velocity or an online Physics classroom at http://www.physicsclassroom.com/Class/1DKin/U1L1d.html for information. The knowledge level is 9th grade.

Discussion and Sharing: (This note is contributed by Jane A. Smith on June 2008) The formula used in this service will be able to serve 80% of applications adequately. However, if a user wants her robots for competition, it is better to use other services such as Velocity7.

Traditional education teaches each STEM subject separately. For example, when teachers teach algebra, they focus on algebraic concepts, such as variables, values and manipulation of values, but barely reference other fields like physics. On the contrary, the students need to apply knowledge from various fields (such as algebra, geometry, mechanics, and game theory) together to build a robotic game successfully. Furthermore, students solve problems during the software design, and the game drives them to learn and be creative. Thus, students reinforce various STEM concepts while learning SOC and playing robotic games. Table 7 shows specific STEM knowledge that can be embedded in common robotic games.

Table 7. Common STEM Concepts that can be Embedded in Robotic Games

Concepts	Detailed Concepts	Way of Introduction
Mechanics (physics)	Velocity, speed, routing, distance, (circular and others) motion, energy, and time.	Robotic programming and games, such as maze travel, knowledge is embedded in services stored in repositories.
Geometry (mathematics)	Geometry such as shortest distance, circles, radius, angle, rotation.	Robotic programming and games such as robot routing. Knowledge will be embedded in services stored in repositories.
Algebra (mathematics)	Variables, values, comparison, manipulation of values	Robotic programming. The knowledge is embedded in services stored in repositories.

Course Details

The proposed SOC course teaches the core concepts and principles of SOC (rather than focusing on tools and languages), including logic of computing, service discovery, workflow, and application composition using services, and their applications in e-business, gaming, and robotics. As a secondary goal, the course covers the latest technologies, such as C# and ASP.Net, or WebSphere and Java, as vehicles for learning SOC. Other relevant but advanced SOC topics, such as security, policy management, information services, and enterprise service, will be covered in junior/senior classes when students have gained relevant knowledge. Table 8 shows a course plan.

The primary goal of the course is to teach students that software construction is mainly a composition of reusable services. The central knowledge is on the ability to discover the needed services including queries and searching in repositories, understand the overall SOC software construction framework, and compose applications by reusing or constructing workflow templates with reusable services. Software construction by reusing workflow template is a new SOC concept developed by the authors (Tsai, 2005). The objectives/expected outcomes of the proposed course are:

- Students will understand the principle SOC concepts including service-oriented architecture, the roles of application builders, service specifications, workflow modeling and specification, service providers and services brokers, dynamic service discovery, basic ontology, and runtime system composition; understand major paradigms of computing: imperative, object-oriented, and service oriented; understand the concept of abstraction.
- Students will be able to specify problems as an application builder: understand a SOC-based modeling language; understand a specific application domain, such as e-business; understand how to decompose the problem in an application domain into required services; apply the language and tools to specify a problem consisting of services and the service flows; bind a remote service into the application specification.
- Students will be able to program simple services as a service provider: understand the basic constructs of an object-oriented programming language; apply a programming language (such as Java or C#) to write simple programs; apply SOC tools to convert a program into a service, with the standard interfaces automatically added.
- Students will understand how to use a service directory: be able to lookup a service directory to find a service; publish a service in the service directory.

Course Schedule

The proposed course lasts 15 weeks as shown in Table 8

Table 8. Course Plan

Weeks	Lecture Topics	Hands-on Laboratories
1	Basic concepts of computing, programming paradigms, computers and computer network from a user's point of view.	Alice game programming illustrating the logic and concepts of workflow composition, drag-and-drop.
2 3	Basic control constructs, data flows, and selection between components and logic.	Alice game programming using basic constructs such as sequential, parallel actions, and conditions.
4	Basic control constructs and logic, designing services as a service provider.	Alice game programming using more constructs such as loops.
5 6	Introduction to SOC, service definitions, service interface, remote access to services, importance of services to control, SOC data routing on Service Bus.	C# programming and service composition using C#. Creating Graphic User Interface (GUI).
7 8	Introduction to an application domain: e-business, introduction to Amazon services and construction of an online bookstore.	Creating a messenger services using service-oriented computing concepts.
9	Introduction to service repositories and service discovery.	Put together sample application, from service programming to application building using services.
10 11	Introduction to Robotics Studio and service composition in VPL.	Microsoft Robotics Studio and VPL, Simulation environment and using LEGO Mindstorm NXT. robots
12	Introduction to service directory and Robotics Studio service repository.	Microsoft Robotics Studio. Programming robotics application of the simulation environment.
13	Writing services as a service provider in VPL and add the services to a service directory.	Using VPL to write reusable drive-control services and deploy the services into Robotics Studio. Programming real LEGO NXT robots.
14	Application integration and testing.	Robot and program testing
15	SOC workshop and ceremony.	Robotics programming and robot competition.

Sample Course Projects

To make the course interesting, so high-school students can learn SOC while enjoying the course, we designed several interesting yet challenging projects and competitions for them (Tsai, Chen, et al., 2008). Aside from the projects mentioned in the previous sections, we held a ball collection and maze traversing competition at the end of the semester for students. The robots used for the competition are LEGO NXT Tribots. The programming environment is Microsoft Robotic Studio 1.5. We provided students with services, such as control dashboard, drive control, and claws control, for them to construct the program they use in the competition.

Competition 1: Ball Collection

Students are divided into four teams. For each round, one team competes with one other team. After 5 minutes, the group with more balls on its side wins the round.

To win in this competition, the students not only need to know how to program with services but also find a balance between making the robot move quickly and making it easy to control.

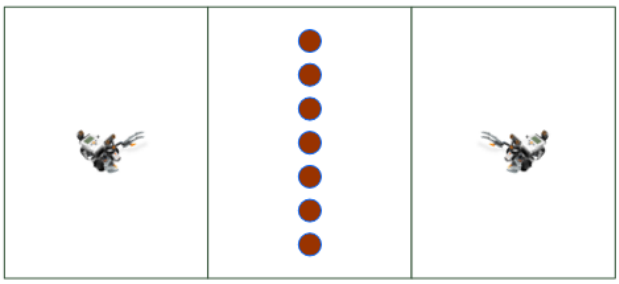


Figure 5. Ball Collection

Figure 5 is a sample application for competition 1. Notice that programming under SOC is really just composing existing services together by dragging and dropping.

Competition 2: Maze Traversing with Artificial Intelligence

Students are divided into four teams. Each team programs their own robot. They are provided with services such as sonar sensor service, notifying the robot the space in front of it, and drive control service that provides turning left and right, 180 degree functions. A suggested algorithm is whenever the robots encounters an obstacle, first check its space on the left hand side, then check on the right hand side, compare the two readings, and move in the direction that has more space.

The maze used in the competition is shown in Figure 6. After 5 minutes, the team wins corresponding points if their robots are in the positions marked in the figure.

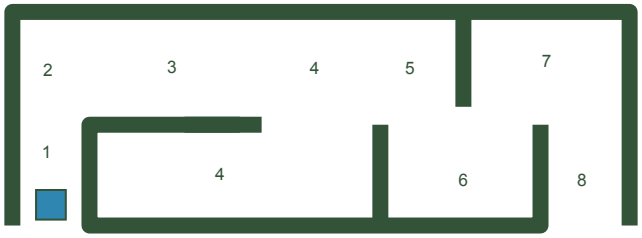


Figure 6. Maze Traversing

Competition 3: Sumobot Competition

Students are divided into different teams. Each team programs their own robot. They are provided with basic services such as those listed in Table 5. In each round of the competition, two teams place their robots in the middle of an arena. Their objective is to push the opponent robot out of the arena.

Pilot Teaching and Evaluation

Based on the approach outlined in this paper, we taught an experimental SOC at Coronado High School in Scottsdale, Arizona in Spring and Fall 2007. The two classes enrolled 28 students and 4 teachers. High-school students and teachers from Scottsdale Unified School District (SUSD) were able to learn the concepts and create SOC applications in e-business and robotics. The course was also offered in summer programs in 2006 through 2008. Since Spring 2008, Coronado High School, Grand Canyon Preparatory, South Mountain High School, and a number of other schools in Arizona also started teaching the course. The course materials can be obtained at the site (SRLAB, 2008c, 2008d).

Purpose of the Evaluation

The evaluation addressed the effectiveness of the project in meeting its goals. The primary goal of the project evaluated was the design and dissemination of computer science curricula and materials aligned to the industry SOC principles and practices. The materials were expected to advance students and faculties' SOC knowledge and application skills. The evaluation assessed (1) the quality of project educational materials and tools and (2) the acquisition of SOC knowledge and application skills by all participants in the project, including pre-service teachers, in-service teachers, and high school students.

The secondary goal of the project was to enhance students' interest in computer science and related jobs. The evaluation measured increases in all participants' (3) interest in computing, more specifically the latest SOC paradigm, (4) motivation for further learning/studies (taking more courses) in computing, and (5) motivation for pursuing a career in computing. The evaluation also assessed (6) teachers' preparedness for delivering SOC courses after completing the undergraduate and graduate SOC courses and workshops. In addition, the evaluation examined (7) the institutionalization of the SOC program at the target institutions and (8) the extent to which SOC materials dissemination was accomplished by this project.

Evaluation Questions

To evaluate the progress made by students, the following eight questions were asked at the beginning and the end of the course. Students were directed to answer the questions by selecting numbers on a scale that corresponded to their proficiency. Here is the translation of the numbers:

3 very well; 2 pretty well; 1 a little bit; 0 not at all

1. I understand and can apply a programming language to write simple programs.
2. I understand and can apply a programming language in problem solving.
3. I understand and can use a service-oriented visual composition language.
4. I understand a specific application domain, such as e-business or robotics.
5. I can decompose a problem in an application domain into smaller problems.
6. I can use a software development tool to design an application.
7. I can convert a program I have written into a service that can be used by other programmers over the Internet.
8. I can search a service directory to find a service.

The following seven questions were asked *after* the course to evaluate the students' assessment of the course and their interests in computer science and SOC.

1. How easy was it for you to understand the material presented?
2. How much did you enjoy the course?
3. How exciting is it for you, personally, to do SOC work?
4. How useful was the course to you, personally?
5. How interested are you in taking more SOC courses?
6. How interested are you in doing a computing major in college?
7. How interested are you in a computing career?

Evaluation Matrix and Results

Table 9 and Table 10 summarize the evaluation results for the multiple choice questions of the two sets of questions respectively from the Spring 2007 offering. A total of 10 students responded to the post-survey during Fall 2007. The rows are the questions and the columns are the choices to each question. The numbers in the tables show the percentages of selecting choices by the participants before and after the class.

Table 9. Results to First Eight Questions Before and After the Course

Choices	**Choice 1 Very Well**		**Choice 2 Pretty Well**		**Choice 3 A little bit**		**Choice 4 Not at all**	
Questions (full questions above)	before	after	before	after	before	after	before	after
1. Write simple programs	0	18.2	18.2	54.5	9.1	27.3	72.7	0
2. Problem solving	0	18.2	0	45.5	36.4	36.4	63.6	0
3. Service-oriented visual language	0	27.3	0	45.5	0	27.3	100	0
4. Specific application domain	0	36.4	18.2	45.5	27.3	18.2	54.5	0
5. Decompose a problem	9.1	36.4	9.1	27.3	18.2	27.3	63.6	9.1
6. Software development tool	0	18.2	18.2	45.5	18.2	36.4	63.6	0
7. Convert a program into a service	0	27.3	0	36.4	9.1	27.3	90.9	9.1
8. Search a service directory	0	27.3	0	27.3	9.1	36.4	90.9	9.1

Figure 7 illustrates the sharp differences between before and after taking the course.

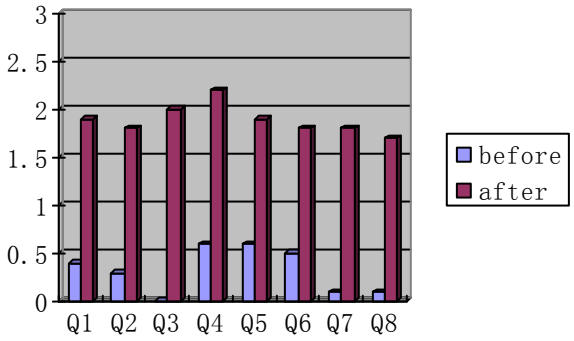


Figure 7. Comparison before and after the Course

From the figure, it is clear that students responded that they learned a great deal in computing and SOC, specifically the average pre-class survey is 0.4 while the average post-class survey is 1.8.

Table 10 Results to Seven Questions After the Course

Choices	Choice 1 Very Well	Choice 2 Pretty Well	Choice a little bit	Choice 4 Not at all
Questions (full questions above)	after	after	after	after
1. Easy to understand the material	0	36.4	45.5	18.2
2. Enjoy the course	27.3	45.5	27.3	0
3. Exciting to do SOC work	18.2	54.5	27.3	0
4. How useful was the course?	45.5	18.2	27.3	9.1
5. Interested in more SOC courses	18.2	18.2	54.5	9.1
6. Interested in computing in college	45.5	9.1	27.3	18.2
7. Interested in a computing career	45.5	9.1	27.3	18.2

Table 10 shows that most students that took the class gave positive feedback and would consider pursuing computer science, specifically SOC as their career.

Lessons and Challenges

We learned many valuable lessons. First, many students are not interested in computer science as their career path. They are interest-driven instead of career-driven. They may be deeply engaged in one activity but may be completely disinterested in another activity. It was a challenge to try to teach them in a systematic and coherent way in an academic course. The course is designed to get them excited at the beginning by teaching them component-based game design and at the end by teaching them robotics application composition. They were indeed excited. We embedded the serious academic contents, such as logic behind computing, binary numbers, and constructs that logically link the components together. We had a hard time maintaining student concentration during this part. However, teachers in the classes had no problem with this part and appreciated the explanations and background knowledge.

Second, due to their heavy teaching loads, few high-school teachers signed up for the course in spring 2007, and only two of them made it through the class. However, during the summer when they had no regular classes, nine teachers signed up for the summer 2007 class. Only one student missed one lecture in the 4-week course with 5 days per week of instruction, and each day lasted three hours. The summer 2007 course was successful as, at the end of the class survey, 80% of the teachers in the class indicated that they would start to teach all or a part of the materials learned in the course. Sixteen teachers attended the summer 2008 class. Some comments written by the teachers can be viewed at (SRLAB, 2008a).

Third, the game programming framework Alice is component-based, not service-oriented. We could not find a service-oriented game platform to fulfill the mission of teaching SOC in this course. However, the component-based Alice is easy to learn and has many SOC features. We will keep working on this issue to find or to develop better suitable framework.

Fourth, the service-oriented robotics computing framework, the Robotics Studio, is an important step in pushing SOC to this domain. Unfortunately, the framework is so new that it has frequent updates and patches. Although Microsoft promised that the Robotics Studio will have full SOC features, many of them were not yet in place as we used the tool. For example, the service repositories provided by the robotics companies must be preloaded into the framework. No independent repositories are available on the Internet. Furthermore, the composition language VPL has a

graphic drag-and-drop interface and is easy to understand, however, it still requires knowledge of control constructs in a programming language, such as the concept of variables, modification of variables, and the modification of states. Thus, even though it is easier to understand than traditional programming languages, students still need to practice. Another issue of VPL is that its loop structure is implemented by "Goto," instead of by structured constructs, and thus difficult to understand. Fortunately, these can be improved in the future.

Conclusions

Overall, the experimental SOC course was successful since some might have doubted that it was even possible to teach SOC to high-school students without prior computing background. Our experimental classes demonstrated that it was indeed feasible and participants' (both students and teachers) evaluations showed that they are interested in this topic and learned the SOC principles. As discussed in the Pilot Teaching and Evaluation section, the SOC classes can be improved, and it will be a collective effort from tool vendors, high school teachers and students, parents, and researchers to work together to make progress. We will continue to experiment with this class so that class materials can be widely disseminated.

References

Arizona State University. (n.d.). *CSE101 Introduction to engineering design*. Retrieved August 16, 2008, from http://www.public.asu.edu/~ychen10/teaching/cse101

BEA (n.d.). *A platform for service-oriented architecture*. Retrieved May 11, 2008, from http://www.bea.com/framework.jsp?CNT=products.htm&FP=/content/solutions/soa/solutions

Chen, Y., & Tsai, W. T. (2008). *Distributed service-oriented software development*. Iowa: Kendall/Hunt Publishing.

Esposito, D. (2004). *Introducing ASP.NET 2.0.* Redmond, WA: Microsoft Press.

Fjuk, A., Karahasanovic, A., & Kaasboll, J. (2006). *Comprehensive object-oriented learning: The learner's perspective.* Santa Rosa, CA: Informing Science Press.

Gillard, S., Bailey, D., & Nolan, E. (2008). Ten reasons for IT educators to be early adopters of IT innovation. *Journal of Information Technology Education*, *7*, 21-33. Retrieved from http://jite.org/documents/Vol7/JITEv7p021-033Gillard257.pdf

Hasan, J., & Duran, M., (2006). *Expert service-oriented architecture in C#: Defining web services development with ASP.NET and WSE 3.0*. California: Apress.

Heinemann, F., & Rau, C. (2003). *Web programming with the SAP web application server*. Maryland: SAP Press.

HP. (n.d.). *HP SOA Center*. Retrieved November 16, 2008, from https://h10078.www1.hp.com/cda/hpms/display/main/hpms_content.jsp?zn=bto&cp=1-11-130-27_4000_100__

IBM Research. (2006). *Services sciences, management, and engineering*. Retrieved April 21, 2006, from http://www.research.ibm.com/ssme

Jadud, M. (2003). *My first programming language*. Retrieved May 16, 2008, from http://www.csed.org/blogs/mjadud/archives/000244.html

Johnson, M. & Liber, O. (2008). The personal learning environment and the human condition: From theory to teaching practice. .*Interactive Learning Environments*, *16*(1), 3-15.

LaPlante, A. (2005). *Education key to SOA success*. Retrieved August, 2008 from http://www.informationweek.com/blog/main/archives/2005/11/education_key_t_1.html

Lohr, S. (2006, April 18). Creating the jobs of the future. *The International Herald Tribune*. Retrieved April 21, 2006, from http://www.iht.com/articles/2006/04/17/business/services.php

Microsoft. (n.d.). *eScience health and wellness initiative*. Retrieved June 8, 2008, from http://research.microsoft.com/ur/us/escience/

Microsoft MSDN (n.d.). *VPL introduction*. Retrieved November 16, 2008, from http://msdn.microsoft.com/en-us/library/bb483088.aspx

Nagel, C. (2005). *Enterprise services with the .NET framework: Developing distributed business solutions with .NET enterprise services*. New York: Addison-Wesley.

Oracle. (2007). *Oracle SOA Suite*. Retrieved May 11, 2008, from http://www.oracle.com/technologies/soa/soa-suite.html

Paul, R. (2005). DoD towards software services. *Proceedings of the 10th IEEE International Workshop on Object-oriented Real-time Dependable Systems*. Los Alamitos, CA: IEEE Computer Society, 3-6.

Phelps, J., & Busby, B. (2007*).* Service-oriented architecture--What is it, and how do we get one? *EDUCAUSE Quarterly*, *11*(3), 56-58.

Resnick, S., Crane, R., & Bowen, C. (2008). *Essential Windows Communication Foundation (WCF).* Addison-Wesley.

SAP. (n.d.). *Enterprise service-oriented architecture.* Retrieved May 11, 2008, from http://www.sap.com/company/press/factsheets/esoa.epx

Singh, M. P., & Huhns, M. N. (2005). *Service-oriented computing: Semantics, processes, agents.* West Sussex, England: John Wiley & Sons.

SRLAB. (2008a). *Comments written by teachers*. Retrieved August 16, 2008, from http://asusrl.eas.asu.edu/ychen/www/activities/robotCamp/CommentCard.JPG

SRLAB. (2008b). *ISOC*. Retrieved May 15, 2008, from http://asusrl.eas.asu.edu/highschool/soc/2007

SRLAB. (2008c). *Robotics.* Retrieved August 16, 2008, from http://asusrl.eas.asu.edu/srlab/Research/Robotics.html

SRLAB. (2008d). *Robotics camp*. Retrieved August 16, 2008, from http://asusrl.eas.asu.edu/srlab/research/roboticscamp

Sun Microsystems. (n.d.). *Sun Java System Active Server Pages 4.0*. Retrieved November 16, 2008, from http://www.sun.com/software/chilisoft/

Tsai, W. T. (2005). *Service-oriented system engineering: A new paradigm.* In Proceedings of the IEEE International Workshop on Service-Oriented System Engineering. Los Alamitos, CA: IEEE Computer Society, 3-6.

Tsai, W. T., Cao, Z., Wei, X., Paul, R., Huang, Q., & Sun, X. (2007). Modeling and simulation in service-oriented software development. *Simulation Journal*, *83*(1), 7-32.

Tsai, W. T., Chen, Y., & Sun, X. (2007). Designing a service-oriented computing course for high schools. *ICEBE: IEEE International Conference on e-Business Engineering*, Hong Kong, 686-693.

Tsai, W. T., Chen, Y., Sun, X., Cheng, C., Bitter, G., & White, M. (2008) Service-oriented computing, International Society for Technology in Education. *ISTE*, *35*(7), 28-30.

Tsai , W. T., Sun. X., Chen, Y., Huang, Q., Bitter, G., & White, M. (2008) Teaching service-oriented computing and STEM Topics via Robotic Games. *ISORC*, 131-137.

Wikipedia. (n.d. a). *Business process execution language*. Retrieved May 11, 2008, from http://en.wikipedia.org/wiki/BPEL

Wikipedia. (n.d. b). *Shared source common language infrastructure*. Retrieved May 11, 2008, from http://en.wikipedia.org/wiki/Shared_Source_Common_Language_Infrastructure

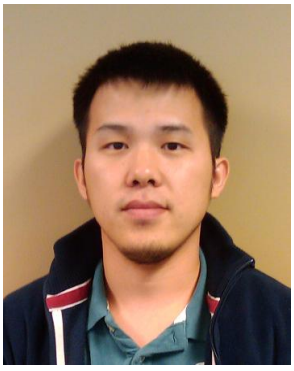
Biographies

Wei-Tek Tsai received his Ph.D. (1986) and M.S. (1982) in Computer Science from University of California at Berkeley, CA, and SB (1979) in Computer Science and Engineering from MIT, Cambridge, MA. He is now a professor of Computer Science and Engineering at Arizona State University. Before coming to Arizona, he was a professor of Computer Science and Engineering at University of Minnesota.

His main research areas are service-oriented computing, software engineering, embedded system development.

Yinong Chen received his Ph.D. from the University of Karlsruhe, Germany in 1993. He was a research fellow at LAAS-CNRS, France, in 1994. From 1994 to 2000, he was a faculty at the University of the Witwatersrand, South Africa, and joined Computer Science and Engineering Department at Arizona State University in 2001. His research areas include computer science education, service-oriented computing, robotics, and dependable system design and analysis.

Calvin Cheng received his Bachelor and Master degrees in Computer Science at Arizona State University in 2005 and in 2007, respectively. His research involves service oriented computing and its applications in high school course curriculum. Currently, he is working at Banner Health as a Software Integration Analyst and is a Faculty Associate at Arizona State University.



Xin Sun received his bachelor degree from Peking University in 2006. He is currently a Ph.D. student in the Computer Science and Engineering Department at Arizona State University.

Gary Bitter is Professor of Educational Technology at Arizona State University and Executive Director of Technology Based Learning & Research (TBLR). He has received lifetime achievement awards from the International Society for Technology in Education (ISTE) and the National Council of Teachers of Mathematics (NCTM) as well as outstanding alumnus awards from Kansas State University and Emporia State University. His groundbreaking research and development of digital curricula and professional development materials is poised to transform PreK-16 and adult learning environments. At Arizona State University, his research focus is on such areas as the impact of integration of technology into curricula on student achievement and attitude, the impact of information technology fundamentals on the digital divide and the future of technology. He has developed several interactive multimedia programs for the professional development of teachers, including Understanding Teaching, Math•ed•ology™ and the ASU-NETS Digital Video Library (DVL). In addition, the e-Learning Network, which is a series of interactive modules designed to help under-advantaged adults learn Internetworking skills and the Hispanic Math Project (HMP), which is an interactive English-Spanish Mathematics program for elementary-middle school students. Bitter earned his Ph.D. in Mathematics and Computer Education at the University of Denver.

Dr. Mary White is currently a Senior Research Specialist for Technology Based Learning and Research. Her research interests include gender issues in STEM precollege education.