Received: 10 May 2017 | Accepted: 21 September 2017
DOI: 10.1002/cae.21885

RESEARCH ARTICLE

WILEY

SEED: A software tool and an active-learning strategy for data structures courses

Marco Adarme[1] | **Daladier Jabba Molinares**[2]

[1] Faculty of Engineer, Department of Systems e Informatics, Universidad Francisco de Paula Santander, Cúcuta, Norte de Santander, Colombia

[2] Department of Systems Engineering, Universidad del Norte, Barranquilla, Atlántico, Colombia

Correspondence
Marco Adarme, Faculty of Engineer, Department of Systems e Informatics, Universidad Francisco de Paula Santander, Cúcuta, Norte de Santander, Colombia.
Email: madarme@ufps.edu.co

Funding information
Universidad Francisco de Paula Santander

Abstract
SEED is a software tool designed for Java consisting of a class library and a set of simulators to facilitate learning of the main data structures. The educational component is based on an active case-solving methodology accompanied by a pedagogical strategy. This strategy allows for the development of a multilayer-model-programming work for the construction of basic and advanced applications in real domains and for integration of its use, development and operation assessment with data structures. The evaluation results of SEED as a pedagogical mediator in the issue of binary trees is presented. The evaluation indicates that students prefer to use SEED due to its simplicity and attractive GUIs which facilitate data structures learning.

KEYWORDS
animation, class library, data structures

1 | INTRODUCTION

To efficiently solve algorithmic problems requires heuristic resolution strategies based on low consumption of critical computing resources (memory & processing) [8]. For this reason, data structures are among the pillars of efficient-applications development [15], and are essential to the optimization of computational resources. Structuring the data for problem solving is an advantage when some information is required. Data can be organized in hierarchical sets so that accessing and processing are as efficient as possible, and optimizing efficiency does not result in design problems or complex coding.

From the beginning of the learning to program, students acquire skills to solve a problem in a given domain and learn to use basic containers (structures). They also take already created components from different languages and use their API in which the implemented operations are insert, delete, update, and query. The issue of data structures has become a compulsory subject in all academic programs related to computing, and has been highlighted as an important approach in curriculum 2013 [23] which comprises a guide to creating content for training courses and programs in the area of information technology and computing. This curriculum emphasizes the study of data structures alongside algorithms and complexity (AL) as well as the importance of being able to use containers and how to implement these data structures in order to solve problems in the field of artificial intelligence, databases and distributed systems, among others. These solutions require efficient processing and data storage.

Currently, the information from all these data structures may not be located in one place, instead it is scattered in many information resources such as books, digital documents, and internet resources that do not allow for managing just a single concept. Additionally, contextualization of these data structures in real domain problems seems to be scarce. Documented sources are focused on the implementation of the structure in very specific programming languages; its explanations are given through abstract models representing memory management based on rectangles or circles, resulting in a conceptually accurate but ambiguous representation for students who conceive the utilization and usage of external structures to performance a formal application. As a result, when and which data structures must be used for the development of an application are confusing.

SEED has been created as a solution to address the need for learning strategies for the use of data structures and their implementation. This software platform includes a class library, which models data structure developed in Java programming language, a documentary resource based on the use of structures in real domain contexts, and pedagogical mediators illustrating the basic primitives of the structures, as well as new implementations developed by the student. For this, SEED is based on the following aims:

- To create an open source component in JAVA for the management and implementation of data structures.
- To develop a working model for the construction of basic and advanced applications in real domains that integrate the use and creation of operations on data structures.
- To develop pedagogical mediators for the manipulation of structures, their basic primitives and new features.
- To provide a system based on conceptual model classes and extended documentation resources using basic design patterns.

SEED was created and used in third-semester Data Structures and fourth-semester Analysis of Algorithms courses, both developed at Francisco de Paula Santander University (Cucuta, Colombia).

2 | RELATED WORK

The implementation of digital resources and libraries in the area of data structures is not something new; however, the detailed analysis of common features of different approaches leads to a better understanding of the topic. Examples of common technological solutions integrated into software to facilitate student learning in the area of data structures include resources such as virtual learning objects, educational mediators, simulators, abstract models, and code visualization of individual structures. To better understand related work and the context in which SEED was conceptualized and developed, we first present an overview of systems that integrate resource visualization and code management software, documentation, class libraries, as well as resources that offer some didactic interaction for learning about data structures.

The Standard Template Library (STL) is an outstanding example of a class library resource and is described in more detail in the CPP Reference guide [5] and by Musser, Derge, and Saini in their STL tutorial [18]. STL is a collection of generic data structures and algorithms written in C++. STL has been adopted by the C++ ANSI standardization committee, which means that it is supported as a language extension for all compilers. This makes STL a great tool for studying and learning different data structures. Similarly, the Aleph-w [13] project of Los Andes University(Mérida-Venezuela) is a library for data structure management in C and C++ with a component for managing frontend graphics that allows for operation with different types of data structures and computational geometry problems.

Data Structure Visualization [9], from the University of San Francisco (USA), is part of the field of didactic visualizations and comprises a series of HTML5 animations with responsive features that can be displayed on any device in standard browsers. Typical computational geometry algorithms and dynamic programming are also included in Aleph-w and the code is open source with clearly defined interfaces, allowing developers to extend the functionality of the frontend.

Another system, CSTutor [4], is an educational software based on the concept of a canvas. In this approach, the student builds a conceptual model of a type of data structure, and the system is capable of transforming it into a C++ source code. CSTutor also works in reverse, building a model from C++ source code. This approach highlights the use of conceptual models that describe the operation of the structure. The CSTutor frontend is IDE-type constructed so that the student can easily integrate the generated code into other applications. Although not a recent project, DSTools [1] stands out as an example for its incorporation of views over applet and documentary support as well as its three-layer architecture. These features result in a basic tool for programming with data structures that allows for graphical debugging and evaluation. The multiple architectural levels in DSTools offer the possibility of easily adding operations on the system, incorporating the concept of modularity, and allowing for delegation and reuse of its class library, an idea that has been exploited for the preliminary design of SEED.

Cupi2collections [24] is a purely academic Java class library with documentary support and some Digital coaches (mediators) that illustrate the implementation of abstract data structures and formal algorithms. This project is part of CUPI2 [25] which includes a pedagogical approach to teaching programming based on problem solving. Cupi2collections main aim is to be a development basis for other structures incorporating the concept of generics and simile parameterization for collections of data structures presented in formal Java libraries (java.util.Collections). Its lead design level are coupling mechanisms easily identifiable through the development of interfaces. It has a hierarchy of data structures that can incorporate mixed containers without difficulty; this design idea is the baseline for creating the SEED classes.

Other projects involve the use of applets to illustrate animations or simple interactions of abstract models, an example being the Java applet Center Center [17], a website containing applets that demonstrate the operation of simple data structures and some fundamental operations. This website consists of animations for sorting and searching algorithms, and it contains the source code and description of each one. The problem in this web page is related to the difficulty in the identification of structures as reusable components. In some cases, authors have developed applets of some data structures in which the process of the implemented operations can be observed in either C++ or Java. In Ref. [3,22], however, only some source codes are completely available, which makes it impossible to work by developing with structures as components.

These published examples describe digital resources for the study of structures as a starting point for building SEED. Now, this research will be focused on three common elements of the analyzed projects: documentation regarding the behavior of data structures, the construction of a class library and a set of simulators for each of the chosen structures, and common data structures that are included in the syllabus of Data Structures and Analysis of Algorithms subjects will be reviewed and the common framework of implementation of the projects discussed.

The common data structures implemented are sequence 20%, linked list 80%, doubly linked list 30%, circular linked list 10%, doubly circular linked list 10% list, queue 90%, priority queue 20%, bi-queue 30%, cyclic queue 10%, stack 80%, binary tree 70%, binary search tree 80%, tree apo 10%, AVL tree 50%, red-black tree 30%, heap 20%, Treap 10%, 10% splay tree, N-ario tree 10%, 10% 1-2-3 tree, 2-3 tree 20%, 10% 2-4 tree, B tree 20%, B+ tree 10%, 74 multitrees 10% quad-tree 10%, 30% graph, and Hash Table 40%. A total of twenty (20) structures are implemented; however, these percentages do not take into account the special implementations, for example, those that provide linear structures with some kind of system or mixed structures. Table 1 shows evaluation items of different projects reviewed with the purpose of showing the strengths and weaknesses of each one, as an innovative development strategy for SEED.

The project with the most implemented structures is STL, and it has become a standard for major implementations on C++. Its advantage lies in separating the native implementation of the structure with operations through the use of generic templates [18]. Nevertheless, Aleph-w and Cupi2Collections offer a class library with a considerable number of developed structures. In regard to teaching tool technologies, the use of Applets stands out. However, some developers use HTML5, and although this view-layer technology is rarely used it would currently be the most recommended for its browser compatibility. Only three (3) projects offer the libraries as a reusable component, and the others just implement their structures as teaching resources.

Taking into account the explicit work of the literature review, SEED is novel in that it offers the combination of a documentary support with a reusable component in any application domain and a graphical tool to evaluate the behavior of structures. Likewise, in the area of academic training for students, its implementation architecture provides a simple view of the real work in a multi-layer environment in which the delegation of operations between the frontend and backend are clearly differentiated.

3 | SEED DESCRIPTION

3.1 | Didactic method

The main purpose of SEED is to allow the student to understand and develop algorithms based on multiprogramming in which he/she can separate the logic of the central exercise (problem situation) from data abstraction and implementation of its operations based on the principles of reuse operations, encapsulation and delegation of operations in an object oriented environment. For this, the student will have four (4) sequential work roles, as shown in Figure 1.

The description of the roles is shown below:

1. Data organizational model analyst (AD): At this point, the student analyzes the appropriate structure to solve a real problem and identifies the operations to be implemented and the interactions between them.

TABLE 1 Comparison of data structures in different projects

	1	2	3	4	5	6	7	8	9
Percentage of implemented data structures	100	90	70	80	70	80	80	60	50
Programming language	C++	C++	Java	Java	Java C++	Java	Java	JavaC++	C++
Documentary resource	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes
Library class as software component	Yes	Yes	No	No	No	Yes	No	No	No
Frontend technology	–	–	Applet	Html5	Html5	Applet	Applet	Applet	Applet

(1) STL, (2) Aleph, (3) DSV, (4) CsTutor, (5) DsTool, (6) Cupi2Collections, (7) Java Applets Center, (8) EdaApplets, and (9) Interactive tutorial data structures.

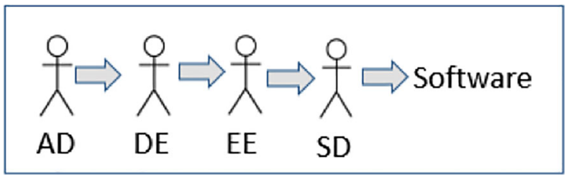


FIGURE 1 Student roles—SEED

2. Structure developer (DE): The student develops operations based on the data structures already created. He understands the use of interfaces and creates new operations.
3. Structure evaluator (EE): Through SEED pedagogical mediators, the student tests the functionality of the created operation, identifies the layered model of the application, and recognizes the delegation between classes of the frontend developed by him.
4. Solution developer (SD): He/she uses the created and evaluated operations in a real domain of work. The student separates the business logic of his/her problem from the implementation of the structure and the corresponding execution, and he/she separates elements within the frontend.

The equation, AD + DE + EE + DS = Software Solution, represents the solution. It is strategically composed by four bodies of knowledge as defined in curricula 2013 [22], which suggests that each course contain multiple areas that should converge not only in the theoretical and laboratory practices, but also in the objectives (targets) to be achieved. These four areas are Software Development Fundamentals (SDF), Discrete Structures (DS), Algorithms and Complexity (AL), and Programing Languages (PL). Table 2 shows the relationship between the teaching work and the area of knowledge that is integrated.

Table 2 shows that the SD role integrates the four bodies of knowledge as previously mentioned. The main idea of working with SEED is to manage the problem of construction, the data-organization understanding and the application domain in each of the data structures, and also the cross-learning understanding of software development through principles and formal object-oriented techniques. As a prerequisite to each content, theoretical sessions should be integrated in order to provide the basis for abstract data types and their main algorithms. Practical sessions include the use of educational mediators provided by SEED and the use of its library in a real problem of formal solution, which aims to consolidate theoretical topics learned and the addition of new operations on the structure and to provide a technological challenge relating to the programming language (addition of a graphical component, reporting, etc.). These four problems are the core of active learning through problems and projects based on the methodology proposed by the Cupi2 project in which students are motivated with the resolution of problems that reflect real-world challenges.

TABLE 2 Student Rol-SEED versus body knowledge

Student Rol-SEED	Body knowledge
AD	DS, AL
DE	DS,AL,PL
EE	DS,PL
SD	DS,AL,PL,SDF

3.2 | Class library

A class library is a set of object classes, made up of special entities such as classes and interfaces (templates), that includes a well-defined interface for their use [12]. The behavior of the classes and interfaces contained in a class library are well-defined and can be reused by different programs [10]. Class libraries may contain classes or interfaces specifically written to function as managers of primary and secondary storage or as elements of frontend and database connections, among others. As a class library contains pre-written, the developer does not have to worry about their implementation and they are an important support tool in solving a programming challenge.

3.2.1 | Analysis and design

Data structures are implemented under the concept of container classes and is based on the design concept in which the structure corresponds to the specification of a set of data and the relations among them [13,18]. In this approach, a specific object class corresponds to a meta level related to the container and in the medium level to turn into a structural composition relation, in which the existence of the contained class (class part) depends on the container class [19]. The class part cannot be shared by other objects, and deleting the container class results in the erasing of its part. Figure 2 shows a typical conceptual model about the construction of each structure within SEED. Each data structure corresponds to a formal taxonomy based on theoretical concepts [6]. SEED takes this conceptualization and designs top-level classes (parent classes) as the starting point for its framework. Figure 3 shows a class-parent conceptual diagram in which the different containers are grouped.

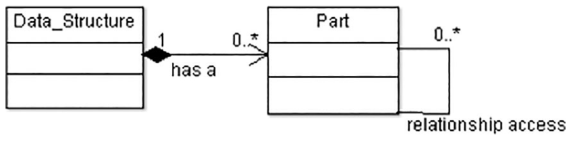


FIGURE 2 SEED—conceptual class diagram

Clustering was designed so that the student can intrinsically identify the application domain concept of each structure. However, the class library is open-ended, and its interface-based reusing mechanisms can easily be changed and operated on another hierarchy. The use of interface allows for the declaration of a set of operations and attributes for classes in order to implement its functional logic. Its main objective is to organize programming and standardize the signing of operations among classes in the same context [14].

The library models two sets of data structures, dynamic and static, respectively. The implementation of static behavior linear structures can be found in the literature. However, for purposes of ease on the learning process, a concept model based on the class diagram in Figure 3 is created. The static denomination is used only for structures of one-dimensional arrays, and the multidimensional abstractions are represented as arrays of arrays with the structure sequence encapsulating this behavior. Figure 4 shows the implemented dynamic structures; its hierarchy and group of top-level classes (called parent class in a generalized inheritance relationship) correspond to a types of data structures [6].

Having explained the previous conceptual model, the diagram in Figure 2 is redefined in inheritance relationships in generalized and specific behaviors.The first is used to share the same attributes (for example, queue class – priority queue class) so that the attributes are factored and placed in a top level class (parent class). The second is for classes classes [7] sharing similar operations (methods), and its logic is implemented by inherited classes. The advantage of using inheritance relationships is that it is easier to reuse code since the attributes and methods do not need to be rewritten and the conceptual hierarchical level between classes is maintained. Figure 5 shows an example section of a conceptual diagram of the SEED tree structure. The following section describes the implementation techniques that were used in Java. The implementation is explained with emphasis on partnerships and heritage that were found in SEED class modeling domain.

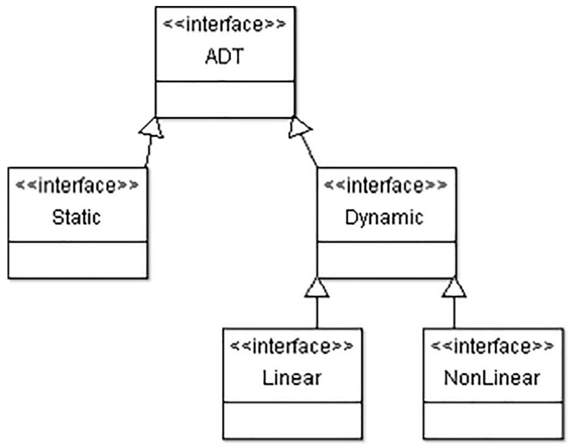


FIGURE 3 SEED—conceptual parent class diagram

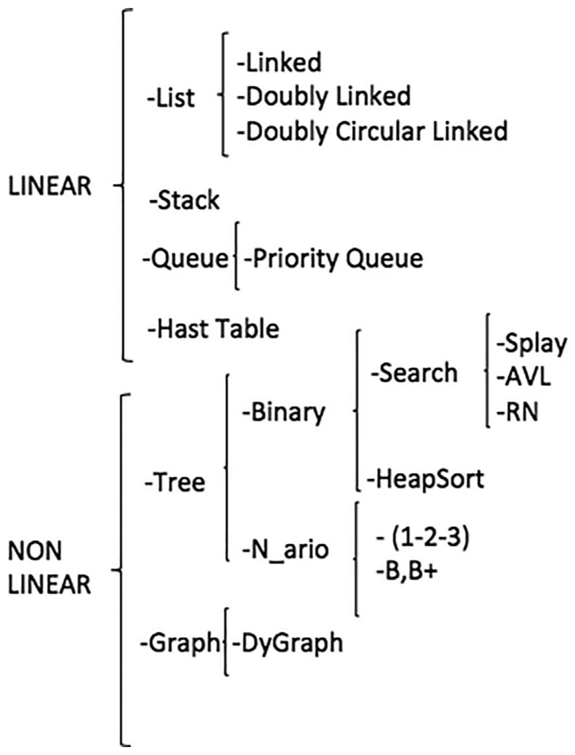


FIGURE 4 SEED—types of data structures

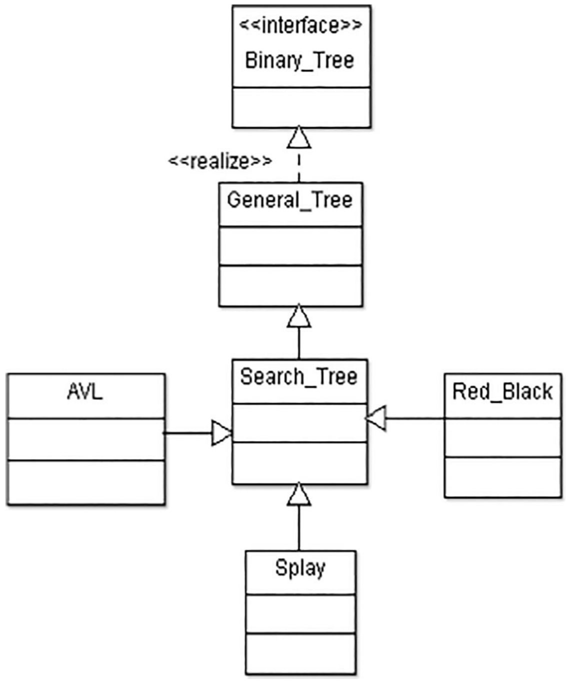


FIGURE 5 Conceptual example diagram of the SEED tree structure

3.2.2 | Implementation

The development of the class library is based on the use of object-oriented design patterns in Java. These patterns contribute to easily reusable classes with a structural cohesion that provides extensibility and flexibility [26] when implementing new operations or structures.

Classes are developed using the Container pattern [20]. Objects of the same type are created in this pattern, and interfaces are available to add, check, delete, and update elements, among other functions. These classes encapsulate such operations, and the structure can be observed as a single entity, usually called collections, in Java. Each Container associates a class with the iterator pattern whose function is to explore the structure and get each of its elements without exposing its internal composition [2,26]. In the case of nonlinear structures, the iterator implements algorithms of particular paths related to the applied structure. An example is binary trees that are traversed in inorder, preorder, or postorder.

The items stored in each container are defined as generic type "T" data [18] a common specification in Java or .Net environments, and they declare a data type to be specified at the time the class is instantiated by using mechanisms of reflection [1]. The concept is also known as "parameterization" as the data type is passed as a parameter to the class to be instantiated. The advantage lies in the standardization of types minimizing the risk of errors at compiling time [21]. Additionally, it works as a recycling mechanism since no behavior code for common elements in the structures would be rewritten, which is commonly seen in the literature where elements of a single type are generally defined as "integer" [11,12,15,16]. Similarly, the reusing allows the created structures to be easily used in any application.

By redefining the diagram on Figure 2, the container is explained in its structural part as a type of generic data associated with an iterator class. Additionally, for the purpose of standardizing the operations on the Java platform and using related-language statements to traverse collections, the iterator and iterable interface and java package (java. langjava.util), are implemented, respectively. This basically defines the contract of use: hasNext(), next(), and remove. In general, the student would eventually create code for these operations within their specific iterator. Figure 6 shows the general class diagram for any data structure. The main package that contains all structures is called "SEED" each container belongs to a class represented by the LabelADT class and corresponds to the specification given in Figure 3, for instance labelADT can be Tree, List, and the others. The Data-Structure class represents the data structure to be treated. However, these can also be derived from another one. For example, in the treatment of trees presented in Figure 5 and the part-class that represents a "node" containing as attribute the information that is stored, called "element," which is a generic data type declared with the character "T." The class My Iterator corresponds to the class that contains the logic path of the structure that implements the standard Java iterators as already mentioned.

3.3 | Pedagogical mediator

The SEED-working model involves the implementation and testing of structures implemented through any Java IDE. NetBeans IDE is preferably taken as a working basis, depending on the context of application of this research, without limiting the activities at the functionality level of other environments. The mediator constitutes a pedagogical and didactic strategy that allows strengthening of the knowledge of each structures behavior with abstract well-defined models through a software simulator. As the simulator automatically develops algorithm animations created in the base structure, this software avoids conventional processes where tests of each algorithm are performed through slides or freehand drawings on blackboards.

The set of simulators implemented in the project, SEED, corresponds to each of the structures previously developed in the classes library. According to the requirements for

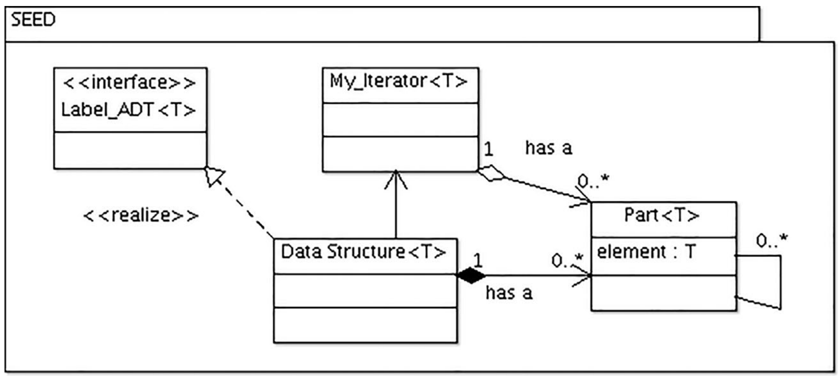


FIGURE 6 SEED—general class diagram for any data structure

graphical representation of each structure, additional components have been used to facilitate student interaction with the graphical simulator. This set of simulators has been developed using a common software architecture (MVC), which makes it possible to separate the presentation layer structures that have functionality from the data that will be presented in the layer. This is done through an intermediate layer having a more appropriate data interface to be displayed for the student. SEED aims to provide an environment to encourage learning where students see the interaction of data structures with different operations that are implemented. The Metamodel package sample diagram of each simulator (see Figure 7) uses the logical architecture of MVC design pattern, encapsulating in each package the management functionality of the structure, the view, and the controller.

The implementation of the set of simulators in SEED is based on starting from a common base for all Simulators, "a structure," that seeks to reduce the implementation complexity in each of them. Thus, the only difference between the simulators is in the data structures for graphical simulation and the graphical presentation of the data structures to the user.

Unlike CsTutor [4] in which the student draws the structure and type based on a modal-window code, SEED can work separately from the structure of the pedagogical mediator. The independence of this component offers the advantage of being easily coupled for use in any real application domain. This fulfills one of the objectives pursued in SEED that not only will the student learn how to implement operations on the structure, but will also apply their behavior and organization to an actual work case. The mediator is therefore a tool to test the concepts discussed in the classroom, which remain a fundamental part in the proposed teaching process. The importance of SEED lies in strengthening work concepts on an architectural model of software based on MVC, allowing for interaction of delegation and Oriented Object design principles, and finally in appropriating the data set as a fundamental element for the good design of efficient programs.

Each GUI mediator was developed in JavaFX, and the interaction consists of creating a button on the simulator to call the controller operation that then calls the operation delegated to the working structure. Assuming that all the simulators will be implemented from the package structure mentioned in Figure 7, the solutions implemented to draw each of the structures using adequate Javafx components remain to be specified. The visualization of each structure gives the student a pleasant and easy interaction in order to strengthen the understanding of the behavior of each one. Thus, a method is implemented with the same signature for each simulator and structures, which allows the drawing and animation of their behavior. This method will be brought to the simulator view after calling the implemented operation by the student, for example the "pintar" TDA method on the GUI TDA class (see Figure 7). This animations encapsulation level and graphical representation allow students not to worry about the management of sophisticated animation algorithms or GUIs, but to focus on the operation that should be

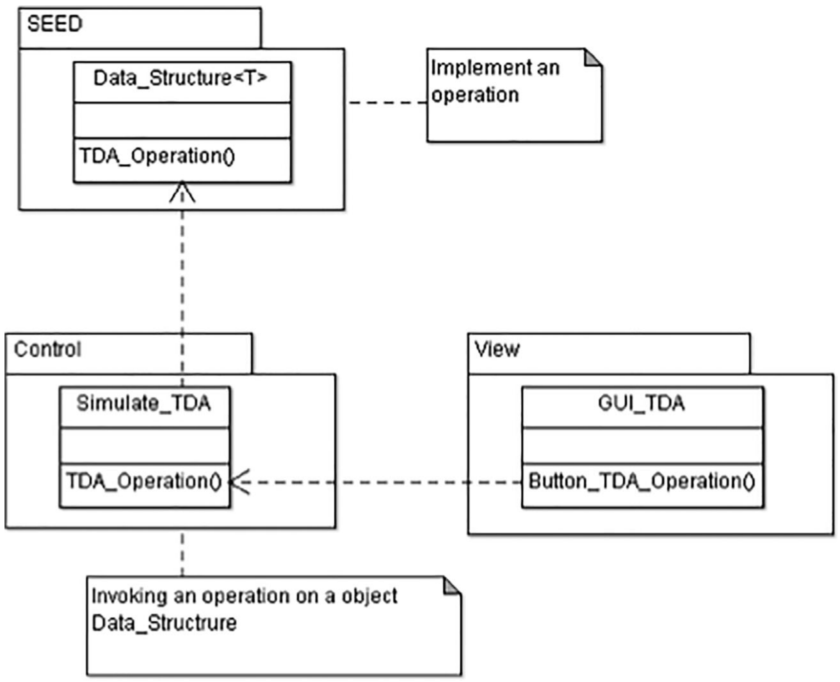


FIGURE 7 SEED simulator-metamodel package diagram

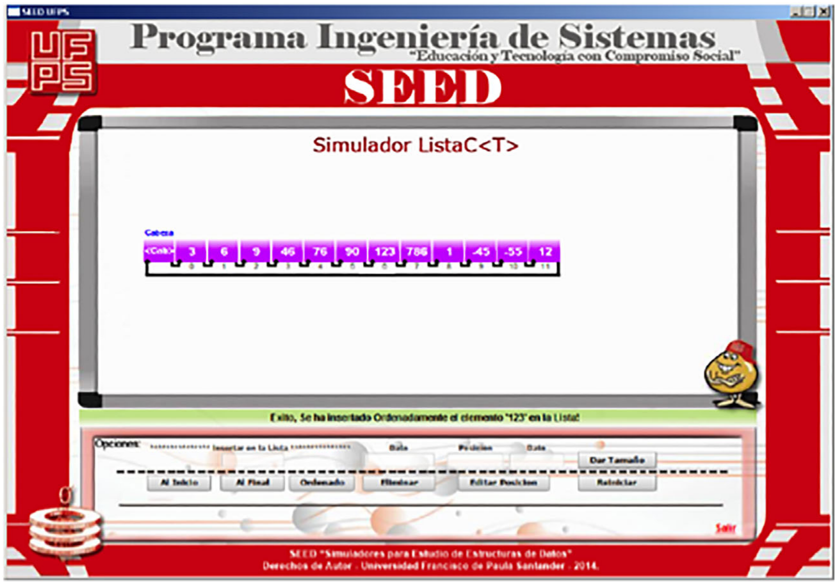

Programa Ingeniería de Sistemas
"Educación y Tecnología con Compromiso Social"
SEED
Simulador ListaC<T>
Exito, Se ha insertado Ordenadamente el elemento "123" en la Lista!
Opciones
Insertar en la Lista
Al Inicio
Al Final
Ordenado
Eliminar
Editar Posicion
Dar Tamaño
Reiniciar
Salir
SEED "Simuladores para Estudio de Estructuras de Datos"
Derechos de Autor - Universidad Francisco de Paula Santander - 2014.

FIGURE 8 A screenshot of linked list simulator

implement or studied about the structure. Figure 8 presents an example of the GUI simulator composed of a drawing area of the structure with a set of buttons, implemented operations, and a space in order to add more Javafx graphics-based components. Each time the student implements an operation, the structure is shown as an animated insertion or deletion of data. If the implemented operation generates some kind of exception, the simulator, for example in the event that any reference of a node fails, presents this result, and a sequential path is not successfully completed.

4 | EVALUATION

To evaluate SEED, two experimental groups were designed. The first group assessed the technical and operational aspects of the class library and its mediator, and the second group assessed the learning effectiveness in exams. During the second semester of 2015, the experiments were conducted with students enrolled in data structures courses. A total of 65 students participated, with 31 students using the SEED platform and 34 students serving as a control group following the traditional methodology. For the study design, a mixed approach was applied such that students were chosen for the section class to which they belonged. The binary trees topic and its variants were chosen as an object of study.

Traditional methodology was defined as that in which the teacher spends most of the time in the master class teaching the concept of the structure, its main operations and its basic implementation in Java with NetBeans with lab time limited to a maximum of two hours. Each student developed his own TDA and made test environments through basic console environments or GUIs with simple inputs, outputs, and text mode. Students who worked with SEED had the same hourly intensity, but followed the previously described SEED learning methodology.

The research study sought to test two hypothesis (Hyp) taken from the experience of the evaluation presented in Ref. [4]:

- Hyp1: Students prefer to work with educational features and tools of SEED than traditional methodology.
- Hyp2: Students obtain higher scores on their tests with the use of SEED.

4.1 | Design of experiment and procedure

The subject under study was binary trees and included the subtopics:

- Standard Tree (general)
- Binary search tree
- AVL

The SEED platform included a didactic teaching (see section 3.1), and its method was generally composed by: a 1 hr master class, 2 hr of laboratory, and 6 hr of independent work. The sub-themes were developed in about 2 weeks per subtopic, as allowed by the course syllabus. The laboratories included interaction with the simulator and the development of methods on the SEED class hierarchy. Each subtopic was evaluated with a laboratory comprising generally of the implementation of methods and the implementation of testing in the simulator through a case study. At the end of each laboratory, students were asked to complete the following survey using a Likert scale, which rates the degree of difficulty for operations within the collections and the simulator (1 = very difficult, 2 = hard, 3 = acceptable, 4 = easy, 5 = very easy):

Q1. Rate the use of SEED to understand and assimilate concepts of binary tree structure data.

Q2. Rate the use of attributes and methods to identify which attributes were located on the binary tree class (SEED Collections).

Q3. Rate the development of methods based on collections presented at SEED.

Q4. Rate the linking to and integration of collection of classes with the class simulator.

Q5. Rate the use of methods called to animate the tree-structure behavior functionality.

Q6. Rate the integration of SEED with NetBeans IDE.

For each of the tests, the student worked with SEED with pre-set scenarios, for example, Figure 9 shows the SEED workspace, options 1 and 2 correspond to the actions buttons for the binary tree algorithms, the buttons grouped in option 2 can be customized by the student, if so desired, to incorporate new algorithms and test them through the animation API created in SEED.In the test it was requested to insert a dataset, this was done in order to verify the correct use of SEED and that the student could answer questions 1-5, mentioned previously.

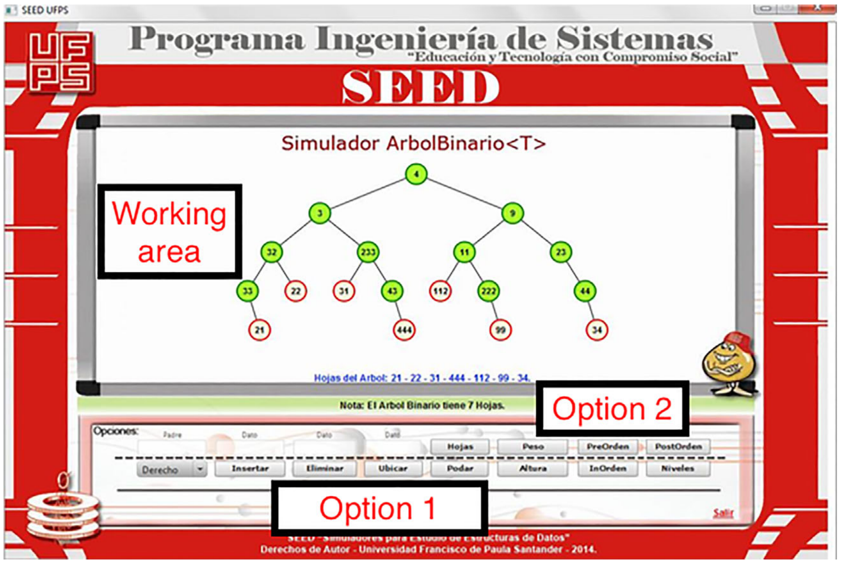
The question 6 was evaluated by analyzing the use of Netbeans with SEED. Figure 10 shows the integration of its components with an IDE is one of the purposes of SEED, for this reason the students must integrate the SEED in Netbeans components. The student has the assignment to identify the multi-layer system presented in the SEED architecture, and the student must finally choose the package and classes to implement new algorithms.

In conclusion, a final evaluation was done using the same exercise for the two groups and results were compared.

5 | RESULTS

Each subtopic was developed in three lab sessions. At the end of each session, the results obtained from the corresponding instrument were collected through an electronic survey. These results are presented in Table 3 as percentages based on the level of acceptance depending on the range of satisfaction. For example 30% of students believed that the use of SEED for structures learning was relevant to their academic process and easy to use. This result was likely due in large part because there was not a pedagogical support tool for the data structure courses. Likewise, the responses for implementation of methods, classroom management and the view-layer simulator integration ease of use, were 26%, 29%, and 33%, respectively. A large percentage of students considered methods to invoke animations difficult to use. This may be because simulators make use of the JavaFX technology for their graphics components, and although the operations are encapsulated, to perform native interpretation methods can be complex, and eventually students considered SEED integration with NetBeans IDE to be acceptable.

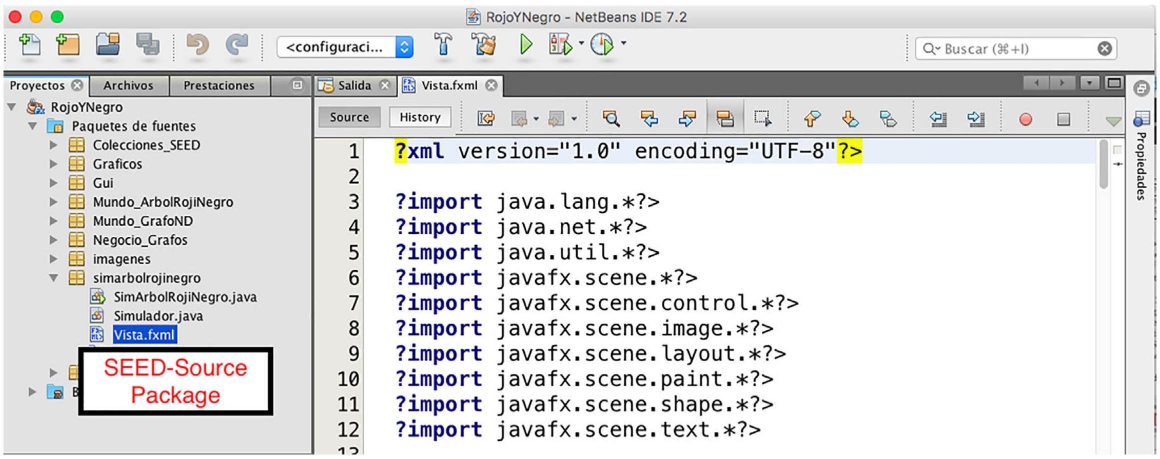
To quantify the effectiveness of SEED compared to a traditional methodology, results from a final exam are compared between the two study groups. Table 4 summarizes the results of the final evaluation as a percentage of students in each group receiving deficient, acceptable, and satisfactory


SEED UFPS
Programa Ingeniería de Sistemas
"Educación y Tecnología con Compromiso Social"
SEED
Simulador ArbolBinario<T>
Working area
4
3
9
32
233
11
23
33
22
31
43
112
222
44
21
444
99
34
Option 2
Option 1
Opciones:
Padre
Dato
Dato
Dato
Hojas
Peso
PreOrden
PostOrden
Derecho
Insertar
Eliminar
Ubicar
Podar
Altura
InOrden
Niveles
Salir
SEED "Simuladores para Estudio de Estructuras de Datos"
Derechos de Autor - Universidad Francisco de Paula Santander - 2014.

Hojas del Arbol: 21 - 22 - 31 - 444 - 112 - 99 - 34.

Nota: El Arbol Binario tiene 7 Hojas.

FIGURE 9 A screenshot of binary tree simulator



```
?xml version="1.0" encoding="UTF-8"?>

?import java.lang.*?>
?import java.net.*?>
?import java.util.*?>
?import javafx.scene.*?>
?import javafx.scene.control.*?>
?import javafx.scene.image.*?>
?import javafx.scene.layout.*?>
?import javafx.scene.paint.*?>
?import javafx.scene.shape.*?>
?import javafx.scene.text.*?>
```

FIGURE 10 A screenshot of SEED in netbeans components

scores. Scoring on the final exam was based on a scoring scale from 1 to 5 according to the following levels of performance:

- Ranging from 1 to 2.9 as a deficient score.
- 3 as an acceptable score
- Ranging from 3.1 to 5 as a satisfactory score

The results show that 42% of students who utilized the SEED approach obtained an acceptable score in the final test (Table 4) while only 36% of students taught with the traditional methodology obtained acceptable scores. However, this difference between acceptable scores in the SEED and the control group was not significant. A possible explanation could be that students who utilized SEED had the option to use a range of algorithms already created in the platform, resulting in easier implementation during the test. While the difference in "acceptable" scores was only minor between groups, a more significant difference was observed in the percentage of students receiving "deficient" scores; 13% fewer students received "deficient" scores in the SEED group compared to the control group. It should be noted that in this experiment, SEED management was applied for only a short period in the class. The next step in this research would be to apply the software tool during the entire semester by integrating SEED into the class syllabus.

6 | DISCUSSION

The results presented in the previous section support Hyp1, that students prefer to work with the proposed methodology SEED in contrast with conventional tools and environments. In fact, working with existing structures facilitates understanding of each of the collections-behavior.

Although SEED was only tested on a part of the course, it was intended as a research strategy applied to learning about binary trees because they include a set of algorithms warranting cases work with other auxiliary structures. This content forced students to analyze the class hierarchy presented in the library and was, therefore, well suited for the implementation of SEED.

On the other hand, even though handling the library with simulators is comfortable, the technology used for animations is complex. Including SEED as educational support for only one of the topics in the class did not represent a problem for the course. However, students did claim that the use of a didactic and easy mechanism is necessary in order to see the structures functionality.

Concerning Hyp2, the results show that the implementation of SEED helps to reduce the failure rate on a particular issue as happens with the binary trees. The success rate (defined as "acceptable"and "satisfactory" performance groups) between the SEED and control groups demonstrates

TABLE 3 Evaluation questions

Likert scale	Q1 (%)	Q2 (%)	Q3 (%)	Q4 (%)	Q5 (%)	Q6 (%)
1	13	16	13	17	13	21
2	20	3	19	13	10	10
3	20	32	10	10	39	28
4	17	23	29	27	16	17
5	30	26	29	33	23	24

TABLE 4 Performance percentage—final exam

Performance	Seed group (%)	Control group (%)
Satisfactory	31	24
Acceptable	42	36
Deficient	27	40

a very small margin of difference. While performance on the evaluation was not drastically different, it is worth highlighting that students during SEED lab sessions remained attentive because the simulator animations were presented through basic methods that had already been implemented in the SEED class library. At the time of testing, students with SEED dynamics work had analyzed a larger number of implemented code in the library.

7 | CONCLUSION

This article presented a tool and a pedagogical strategy, called SEED, proposed for teaching and learning data structures intended for use in engineering and systems-related courses. First, a literature review of current state of the art approaches related to the theme was presented showing the main academic contributions of each approach and evaluating program characteristics that were taken into account when constructing SEED. Then, the SEED educational component was described in detail consisting of three parts: (1) teaching strategies and learning, (2) the class library, and (3) the development and implementation of simulators.

The results of the evaluation show that the software solution and proposed strategies could help as a teaching mediator for teaching/learning the concepts of data structures. Similarly, students who used SEED in their classes showed greater effectiveness in understanding the handling of data structures and their relationship when applied to a real case. Programming SEED simulators allowed students to have a multilayer environment that helped facilitate understanding of conventional software development.

Although there are a number of academic implementations for this purpose, SEED provides an easy-to-use API in Java programming language and graphical interfaces made with a high level of usability with a didactic-pedagogical strategy that simulates quick manual operation of each one of the data structures. The proposed strategy includes the dynamics expressed by active learning styles where the student can see exactly what the abstract modeling of each structure means in relation to each of their behaviors or functionalities.

ACKNOWLEDGMENTS

The first author expresses his deep thanks to the Universidad Francisco de Paula Santander (Cúcuta Colombia) for their academic and financial support. He also expresses his deep thanks to the Universidad Del Norte (Barranquilla Colombia) for their academic support.

ORCID

Marco Adarme http://orcid.org/0000-0002-2121-1208

REFERENCES

1. M. Adarme, *Reflexión computacional un enfoque desde C++*. Cúcuta-Colombia: ECOE EDICIONES (2013).
2. AllAppLabs.com, *JAVA DESIGN PATTERNS*, 2014. [Online]. Available online at: http://www.allappforum.com/java design patterns/iterator pattern.htm
3. L. Almeida, F. Blanco, and V. Moreno, "EDApplets: Una Herramienta Web para la Enseñanza de Estructuras de datos y Técnicas Algorítmitcas Almeida F., Blanco V., Moreno L. M." *X Jornadas de Enseñanza Universitaria de la Informática*, 2003, pp. 1–8.
4. S. Buchanan and J. J. Laviola Jr. *CSTutor: a sketch-based tool for visualizing data structures*, Trans. Comput. Educ. **14** (2014), 3:1–3:28, [Online]. http://doi.acm.org/10.1145/2535909
5. Comunidad C++, "CPP Reference" (2015). [Online]. Available online at: http://es.cppreference.com/w/cpp/container
6. T. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*, MIT Press, Cambridge, MA, 2009.
7. A. Dennis, B. H. Wixom, and D. Tegarden, *Systems analysis and design: An object-oriented approach with UML*. John Wiley & Sons, New York, NY, 2015.
8. A. Duch, *Análisis de Algoritmos*, U. P. de Barcelona, Ed., Barcelona, 2007.
9. D. Galles, Data Structure Visualizations, San Fracisco, USA, 2011. [Online]. Available online at: https://www.cs.usfca.edu/galles/visualization
10. N. S. Gill, *Importance of software component characterization for better software reusability*, ACM SIGSOFT Software Engineering Notes, **31** (2006), 1–3.
11. M. T. Goodrich and R. Tamassia, *Data structures and algorithms in Java*. Wiley, Hoboken, NJ, 2008.
12. T. Groussard, *Java 7: los fundamentos del lenguaje Java*. Ediciones ENI, Cornellà de Llobregat (Espagne), 2012.
13. L. R. León, "ALEPH-w," 2012. [Online]. Available online at: http://www.webdelprofesor.ula.ve/ingenieria/lrleon/aleph/html/index.html
14. T. Lindholm, F. Yellin, G. Bracha, and A. Buckley, *The Java virtual machine specification*. Pearson Education, 2014.
15. L. J. A. Y. I. Z. Martínez, *Programación en C, C++, Java y UML*, McGraw-Hill, México D.F., 2014.
16. W. McAllister, *Data structures and algorithms using Java*. Jones & Bartlett, New York, 2010.
17. R. Mukundan, "Java Applets Centre—Data Structures," 2006. [Online]. Available online at: http://www.cosc.canterbury.ac.nz/mukundan/dsal/appldsal.html
18. D. R. Musser, G. J. Derge, and A. Saini, *STL tutorial and reference guide: C++ programming with the standard template library*. Addison-Wesley Professional, Reading, MA, 2009.

19. A. G. Parada, E. Siegert, and L. B. de Brisolara, Generating Java code from UML class and sequence diagrams, in *2011 Brazilian Symposium on Computing System Engineering*. IEEE, 2011, pp. 99–101.
20. A. Schatten, "Container Pattern," 2013. [Online]. Available online at: http://best-practice-softwareengineering.ifs.tuwien.ac.at/patterns/container.html
21. J. F. V. Serrano, A. P. Abril, F. G. Bellas, and Á. S. Calle, *Diseñar y programar, todo es empezar: Una introducción a la programación orientada a objetos usando UML y Java*. Dykinson, 2010.
22. F. Tejada Maldonado and Santalla Adriana, "Tutorial Interactivo Estructura de Datos," 2011. [Online]. Available online at: http://osiris.ucb.edu.bo/inf104/index html/
23. The Joint Task Force on Computing Curricula (Association for Computing Machinery IEEE-Computer Society), Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science, 2013. [Online]. Available online at: http://ai.stanford.edu/users/sahami/CS2013/strawman draft/cs2013-strawman.pdf
24. UNIANDES, "Proyecto Cupi2Collections," 2012. [Online]. Available online at: http://cupi2.uniandes.edu.co/sitio/index.php/cursos/estructurasde-datos/cupi2collections/
25. J. A. Villalobos and N. A. Calderón, *Proyecto Cupi2: un enfoque multidimensional frente al problema de enseñar y aprender a programar*, Revista Investigaciones UNAD, **8** (2009), 45–64.
26. M. Yener and A. Theedom, *Professional Java EE design patterns*. John Wiley & Sons, Wiley, Oxford, UK, 2014.

M. Adarme is PHD student in System Engineering and Computer Science from Universidad del Norte, Barranquilla, Colombia and is assistant professor and researcher at Universidad Francisco de Paula Santander, Cúcuta, Norte de Santander, Colombia.

D. Jabba Molinares received the PHD degree in Computer Science and Engineering from the University South Florida and is assistant professor and Director of Research, Development and Innovation at the Universidad del Norte, Barranquilla, Colombia.

How to cite this article: Adarme M, Jabba Molinares D. SEED: A software tool and an active-learning strategy for data structures courses. *Comput Appl Eng Educ*. 2017;1–12. https://doi.org/10.1002/cae.21885