Introducing Good Design Principles in an early System Engineering Course

Claudia Pereira
cpereira@exa.unicen.edu.ar

Liliana Martinez
lmartine@exa.unicen.edu.ar

Laura Felice
lfelice@exa.unicen.edu.ar

Martín Meliendrez
mmeliendrez@alumnos.exa.unicen.edu.ar

Facultad de Ciencias Exactas
Universidad Nacional del Centro de la Provincia de Buenos Aires
Tandil - 7000 - Argentina

Abstract

A System Engineering curriculum should involve elements or concepts that reflect a trend towards distinguishing the true software professional from the occasional programmer. In order to introduce some fundamental concepts in an early stage of a System Engineering career, this paper proposes the application of a global project in a second year, where students have to integrate the concepts learned and acquire the necessary skills to work in a professional environment. Also, the used methodology throughout the course, which introduces good design principles early in the career, is detailed. In particular, a global project which puts emphasis on reuse is presented.

Keywords: algorithm design techniques, programming teaching, software reusability, capstone project

1. INTRODUCTION

A System Engineering curriculum should involve elements or concepts that reflect a trend towards distinguishing the true software professional from the occasional programmer. This trend has important consequences for universities. What matters is to teach students fundamental ways of thought that will accompany them throughout their careers and help them grow in this ever-changing field. As Bertrand Meyer analyses in (Meyer, 2001), a software curriculum should involve five complementary elements:

- principles: lasting concepts that underlie the whole field;
- practices: problem-solving techniques that good professionals apply consciously and regularly;
- applications: areas of expertise in which the principles and practices find their best expression;

- tools: state-of-art products that facilitate the application of these principles and practices; and
- mathematics: the formal basis that makes it possible to understand everything else.

We use a methodology based on the points mentioned above to teach in the second-year course of the System Engineering career "Analysis and Design of Algorithms I" at Universidad Nacional del Centro de la Provincia de Buenos Aires (ADA1, 2009). This methodology starts at abstraction level defining the problem domain and identifying the involved abstract data types (ADTs). They are formally specified and organized in libraries to enable their reuse. Abstraction mechanisms are of paramount importance in object-oriented programming (Stroustrup 1997), which is a programming paradigm to define the abstract data types and their relationships.
The problem definition corresponds to clarifying the initial ideas, which in turn, would determine the final outcome of the project. Students are confronted with a concrete problem, and they need to separate necessary from unnecessary details: students try to obtain their own abstract view, or the problem model. This process of modeling is called abstraction.

Finally, at implementation level, the formal specification of data types and the algorithms that use them are implemented in C++ language (Dale, 1998) (Stroustrup 1997). These algorithms are based on problem-solving techniques that are taught throughout the course such as greedy method, divide and conquer, backtracking and dynamic programming.

The problem solution to be defined depends on the student's knowledge about programming languages, algorithms design techniques and available development environments (IDEs).

The learning is mainly based on problem solving. On the one hand, the students solve practical exercises corresponding to each topic of the course. On the other hand, they develop a capstone project based on real situations, which allows students to integrate the learned concepts and thus, prepares them to work in a professional environment. In this paper, we focus on the global project of the course and present a study case which puts emphasis on reuse.

This paper is organized as follows. Section 2 presents the methodology applied in the course. Section 3 describes the course content and the teaching strategies. Section 4 describes practical aspects of the global project. Section 5 describes a study case developed by a student group. Finally, Section 6 considers conclusions.

2. THE METHODOLOGY

The goal of ADA1 is to introduce the following basement for software development: ADT specification, integration of specification and implementation, construction of component library and software reusability. In order to achieve this goal, the proposed methodology starts at abstraction level, which formally describes a problem independently of the data type representations and of a particular programming language, and concludes at implementation level with efficient programs written in C++ language.

At abstraction level, the problem domain is defined and the entities that intervene in the problem are identified. The classes of objects are identified and algebraically specified in NEREUS specification language (Favre, 2006). The fundamental design principles of this specification language are complete and incomplete specifications, genericity, inheritance and clientship. These language features allow the incremental construction of specifications reusing previously specified types for the solution of other problems. The organization in libraries of types and designs facilitates the reuse.

At implementation level, the formal specifications are translated to C++ language code and integrated with algorithms that intervene in the solution of the problem. Throughout ADA1 the students solve practical exercises that use the type library and apply different techniques of algorithm design such as greedy method, divide and conquer, backtracking and dynamic programming.

In System Engineering career, students learn several programming languages that correspond to different paradigms. In the first year of the career, the principles of structured programming are taught and

Pascal is the programming language used. In third year, the course of Object Oriented Programming introduces the basic and advanced concepts of the paradigm. They use Java language. For this reason, in ADA1 (second level), C++ language is used to establish a connection between structured and object oriented programming. C++ is a hybrid object oriented language that allows working with object classes that belong to the problem domain and functions that manipulate them in an independent way.

By applying this methodology, the students are able to:

- acquire a high abstraction level of thinking, distinguishing the essential from the irrelevant in a class specification,
- distinguish specifications from implementations,
- learn, throughout the practices, the ability to decide what information will be hiding (information hiding),
- increase the reuse of types and design to reduce the time and effort required to build software systems.

Partially, this methodology was published in Favre et al. (1998), Favre et al. (2000) and Felice et al. (2002).

3. COURSE DESCRIPTION

ADA1 is a second year course in our University. This first semester course is a nine-hour per week program, three-hour theoretical classes, three-hour practical classes and three-hour computer laboratory practices. The suggested methodology emphasizes active learning via concrete laboratory experiences.

ADA1 provides the students with the fundamental concepts such as computational complexity, recursion, ADT and design techniques of algorithms like backtracking, divide and conquer, dynamic programming and greedy method. The integration of the algorithms with ADT libraries constructed by the students plays a central role in the course. This course is strongly articulated with a second semester course -Analysis and Design of Algorithms II (ADA2, 2009)- which continues with this methodology covering more advanced topics such as graph theory and approximation and geometric algorithms.

Teaching strategies that encourage constructivist learning are used to carry out the proposed methodology. Learning, in a constructivist view, is an active process which involves complex interaction between previous knowledge of the student, social context and the problem to solve (Jonassen, 1999). These strategies are:

- Problems resolution: students solve practical exercises either individually or in groups. This strategy stimulates the connection between learned knowledge and its application in actual and concrete situations.
- Development of a global project: this project, developed in groups of two or three students, integrates the main issues of the course. This strategy encourages interaction among the students who work together to reach a common goal.

The resolution of problems is based on classical exercises that cover the course topics. The first topics like computational complexity, recursive programming and abstract data types give students a solid foundation for further study. The remaining topics, algorithm design techniques, provide students with the conceptual tools to solve problems using computers. The practical problem of implementation in large is addressed in the second part of the semester, when C++ language is used to examine how abstractions are expressed efficiently.

The global project is a large-scale development where the proposed methodology is applied to solve a problem based on a real and concrete situation. In particular, game applications are a proper topic for the development of the project (Felice and Fernandez, 2006). The project allows students to achieve:

- identification of the ADTs that intervene in the problem,
- algebraic specification of the ADTs reusing type libraries and design,
- selection and combination of algorithm techniques to solve the problem, and
- implementation of the whole project.

4. THE GLOBAL PROJECT

The goal of the global project is to provide students with real experience in solving large complex "real world" problems. The focus is the design, production,

documentation and maintenance of the project with particular emphasis on concepts like reusability both in programming and data design.

In ADA1 course, the global project has been implemented for years and its feedback allows improving the project design each time.

The number of students that attend this course is approximately 150. The project is developed within an eight-week time frame by students in groups of two o three.

Each group is assigned a mentor who belongs to the teaching staff of ADA1 course. The mentor advises and helps the groups throughout the project development. The group-mentor contact takes place during the computer laboratory practices.

Project organization: practical aspects

The teaching staff formulates the project taking into account:

- experiences and knowledge of the students,
- the problem to solve must be interesting and relevant to represent a challenge for the students,
- deliverables and established time frame of each project stage should be carefully designed.

Then, the mentors develop the project schedule. Essentially, it is carried out in two stages in which the groups should produce the following deliverables.

Stage 1 - deliverables for the specification: In this stage, the problem domain is described and algebraically specified. Each group has to submit a report with the following contents:

- definition and specification in the NEREUS language of the entities that intervene in the problem making explicit the reused TDAs,
- class diagram that depicts the TDAs and its relationships,
- possible data structures to implement the TDAs,
- analysis of the suitable techniques to solve the problem.

Stage 2- deliverables for the implementation: In this stage, the TDAs and the algorithms that solve the problem are implemented. The deliverables consist of:

- a report containing the implementation details,
- CD/DVD containing source code and executable file

The mentor checks the deliverables of each stage and gives students feedback. In case there were mistakes, students should re-do the deliverables accurately.

Finally, each group presents the deliverable final version of the project to their peers and mentor. Each group member must demonstrate their involvement in the project to get through it.

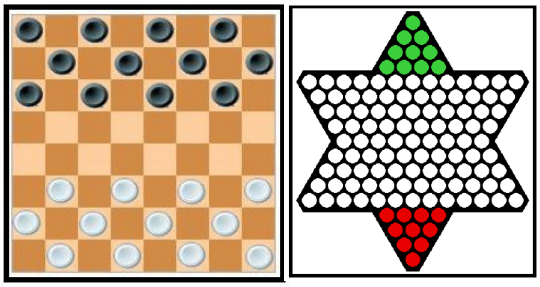
5. A STUDY CASE

In this paper, we partially show a final project developed by a student group of 2 members at the end of the first semester, 2008 (Meliendrez, 2008). They had to implement board games that allow playing human vs. computer. The computer had to play intelligently.

One of the project aims was to construct a design that fulfils the needs of most board games. To find out their common features, students required a thorough understanding of every possible game. The design should be extendible for other board games. However, the implementation was restricted to classical Checkers and Chinese Checkers Board. The classical Checkers game board has 64 squares (Figure 1.a) and the Chinese Checkers board is a six-point-star called David star (Figure 1.b). Both games are board games played between players alternating movements. Briefly, the objective of the Checkers game is to eliminate all opposing checkers or to create a situation in which it is impossible for the opponent to make any movement. Normally, victory will be due to complete elimination. On the other hand, the aim of the Chinese Checkers game is to be the first player to move all ten pegs (pieces) across the board and into the opposite triangle.

Another project aim was to apply algorithm design techniques in order to simulate the computer intelligence to play the game.

Next, deliverables for stage 1, class diagram and the proper techniques to solve the study case problem are partially depicted.



a) Classic b) Chinese

Figure 1. Checker Boards

The Software Design

As Meyer mentioned in (Meyer, 1997), in many cases the abstraction is not unique; how best to abstract a certain notion depends on what the programmer wants to do with the notion and its variants. Consider for example our study case: at least four abstractions are possible.

Figure 2 (see appendices) partially shows the class diagram obtained. The main ADTs are detailed below.

Board: it declares the common methods for any type of board, such as methods to charge the starting positions, to move a piece, to get the position of a piece. Board is an abstract class; thus, particular types of boards specialize such abstract class, and implement methods according to its features.

Checkers and Chinese Checkers Boards: are concrete boards which have their own attributes and implement the abstract methods of Board such as *move* and *insertPiece*.

Piece: each piece has a color and a position in the board. This class declares the common methods *getColor* and *getMovement*. Pieces have particular movements according to the rules of each game. For instance, in the Checkers game there is a distinction between the common piece and the checker movements. Thus, *getMovement* will be implemented by each subclass: ChineseCheckersPiece, CheckersPiece and KingCheckersPiece.

CheckersGame: it is the class responsible for playing. It is made by selecting the strategy corresponding to each game. The first design proposed by students was improved introducing design patterns. This more advanced topic was suggested by the mentor. In order to encapsulate the algorithms corresponding to Checkers and Chinese Checkers games, and to make them interchangeable, the best solution is the behavioral pattern Strategy (Gamma, 1995, pp. 315). Hence, different algorithms were implemented for each game. Figure 3 shows the Strategy pattern and Figure 4 depicts its application for this particular problem (see appendices).

Algorithms design techniques

In order to implement the machine intelligence for each game, different topics given in the course were used. The method *getBestMovement* of MaxMinStrategy class implements the movements of the computer in the classical Checkers game using the backtracking technique. In particular, Min-Max strategy which is applied in two-player games is implemented (Aho et al., 1983).

The method *getBestMovement* of EuclideanDistance-Strategy class implements the movements of the computer in the ChineseCheckers game using a heuristics based on the Euclidean distance algorithm (Cormen et al., 1990).

The implementation

The project implementation was developed in Code::Blocks Integrated Development Environment. The main idea is that this application runs in many platforms, so the used libraries are multi-platform. During the development of this project the C++ GCC (GNU Compiler Collection) was used. For Windows platforms, MinGW (native software port of the GNU) could be used.

6. CONCLUSIONS

This paper presents the application of a global project that integrates the learned concepts throughout the second-year course ´Análisis y Diseño de Algoritmos I´ of System Engineering career. In addition, a teaching methodology that provides students with strong conceptual foundations since the beginnings of the career is presented.

The application of our approach has demonstrated good results. A high rate of students has met the course expectations. On the one hand, by applying the proposed methodology students acquire a high abstraction level of thinking distinguishing

specifications from implementations. Moreover, they increase the reuse of types and design to reduce the time and effort required to build software systems. On the other hand, the development of a global project based on real situations allows students to integrate the learned concepts and to acquire the skills that they need to work in a professional environment.

Since 2004 we have integrated global projects to ADA1 and ADA2 courses. Throughout these years, student's feedback has revealed that both classes of laboratory and mentor guide are essential to carry out successfully the development of the project. Some global projects were published at student's symposiums (Ridao 2005, Defelippe, 2009, Martínez, 2009, Iaruzzi, 2009, Ferrante 2009, Fernandez 2005). Fernandez (2005) obtained an award at EST 2005 symposium. These publications demonstrate the success of our approach.

7. REFERENCES

ADA1 (2009) URL: www.exa.unicen.edu.ar/catedras/aydalgor

ADA2 (2009) URL: www.exa.unicen.edu.ar/catedras/aydalgo2

Aho, A.; Hopcroft, J.; Ullman, J. (1983) "Data Structure and Algorithms". Addisson- Wesley.

Cormen, T.; Lieserson, C.; Rivest, R. (1990) "Introduction to Algorithms" Ed. The MIT Press.

Dale, N; Weems, C. (1998) "Programming in C++", Jones and Bartlett Publishers. ISBN 0-7637-0537-3.

Defelippe, F. (2009) "Algoritmos heurísticos de resolución para puzzles de piezas deslizantes". "38º Jornadas Argentinas de Informática e Investigación Operativa" (38 JAIIO) August 2009. Mar del Plata. Argentine. pp. 346-359. Ed. Silvia Castro, Javier Orozco. ISSN 1850-2946.

Favre, L. (2006) "A Rigorours Framework for Model-Driven Development". In Advanced Topics in Database Research Series. Vol 5. Chapter 1. IGP (Idea Group Publishing). Keng Siau Ed. USA. pp: 1-27.

Favre L.; Felice L.; Martinez L.; Pereira C. (1998) "Análisis y Diseño de Algoritmos: Un enfoque a partir de especificaciones algebraicas". In: "IV Congreso Iberoamericano de Educación Superior en Computación". Entidad organizadora: CLEI. Ecuador 13-16 de Octubre. pp: 19-28.

Favre L.; Felice L.; Martinez L.; Pereira C. (2000) "On Teaching a Data Structures and Algorithms Course through a Rigorous Approach". In ISECON 2000 (Information Systems Education Conference). Philadelphia, Pennsylvania, USA. 7 pages.

Felice L.; Martinez L.; Pereira C. (2002) "A formal approach to the teaching of Abstract Data Types". In: Proceedings of 2002 Informing Science + IT Education Conference. Irlanda. ISSN 1535-0703. pp: 465-472.

Felice, L.; Fernández, M. (2006) "The use of Games to teach Programming Algorithms". In: Proceedings of ISECON 2006: Boot Up IS Education. Nov 2-5, Dallas. USA

Fernandez, M. (2005) "Algoritmos de búsqueda heurística en tiempo real. Aplicación a la navegación en los juegos de video." EST 2005 (Concurso de Trabajos Estudiantiles): 34 JAIIO. Jornadas Argentinas de Informática e Investigación Operativa. Rosario. Argentina.

Ferrante E. (2009) "Clausula: Herramienta Didáctica para la Enseñanza de Lógica de Predicados de Primer Orden". "38º Jornadas Argentinas de Informática e Investigación Operativa" (38 JAIIO) August 2009. Mar del Plata. Argentine. pp. 412-423. Ed. Silvia Castro, Javier Orozco. ISSN 1850-2946.

Iaruzzi, E.; Pereyra, A. I. (2009) "Pathfinding utilizando Algoritmos de Hormigas Aplicado a laberintos 3D" . "38º Jornadas Argentinas de Informática e Investigación Operativa" (38 JAIIO) August 2009. Mar del Plata. Argentine. pp. 447-458. Ed. Silvia Castro, Javier Orozco. ISSN 1850-2946.

Jonassen, D. (1999) "Designing Constructivist Learning Environments". In M. Reigeluth (Ed.) Instructional-design Theories and Models: A new paradigm of instructional theory. Volume II. Pages:

215-239. Mahwah, NJ: Lawrence Erlbaum Associates.

Martínez, C. (2009) "Heurísticas aplicadas a la resolución del problema del Viajante múltiple" "38º Jornadas Argentinas de Informática e Investigación Operativa" (38 JAIIO) August 2009. Mar del Plata. Argentine. pp. 267-280. Ed. Silvia Castro, Javier Orozco. ISSN 1850-2946.

Meliendrez, M. (2008) "Reuso de diseño y código: aplicación a los juegos de damas y damas chinas". Internal Report.

Meyer B. (1997) "Object-Oriented Software Construction". Prentice Hall., Inc. ISBN 0-13-629155-4.

Meyer B. (2001) "Software Engineering in the Academy". In. Computer (IEEE), Vol.34, No.5, May 2001, pp: 28-35.

Ridao Freitas, I; Vidal, S (2005) "Algoritmos de resolución para el cubo de Rubik". EST 2005 (Concurso de Trabajos Estudiantiles): 34 JAIIO. Jornadas Argentinas de Informática e Investigación Operativa. Rosario. Argentina.

Stroustrup, Bjarne (2000) The C++ Programming Language (Special Edition). Addison Wesley. Reading Mass. USA. 2000. ISBN 0-201-70073-5. 1029 pages. Hardcover.

Appendices

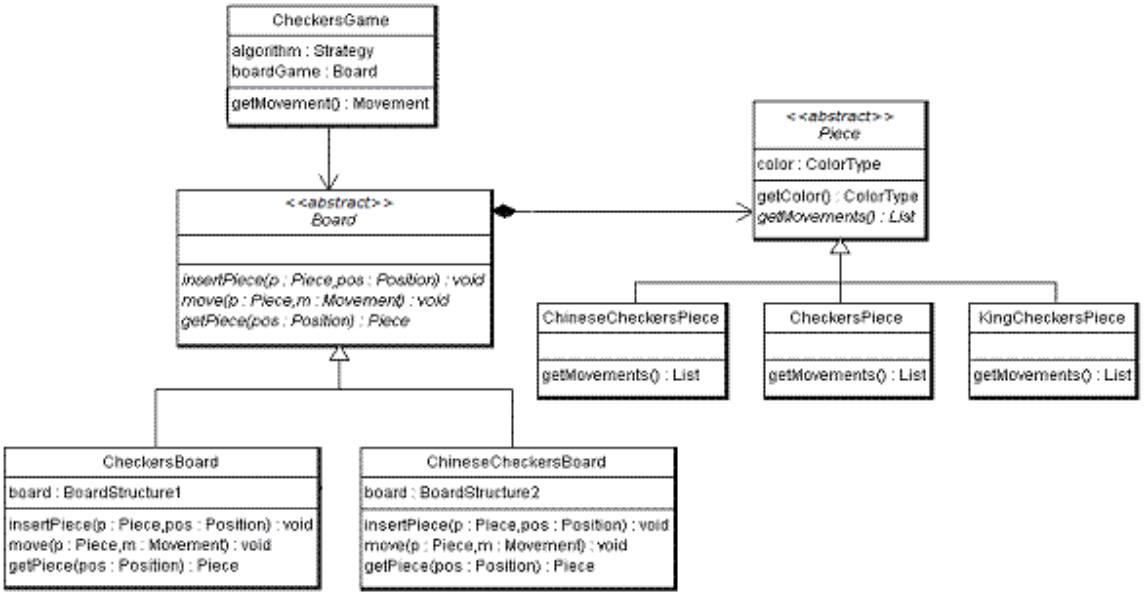


Figure 2. Adaptation of Strategy Pattern

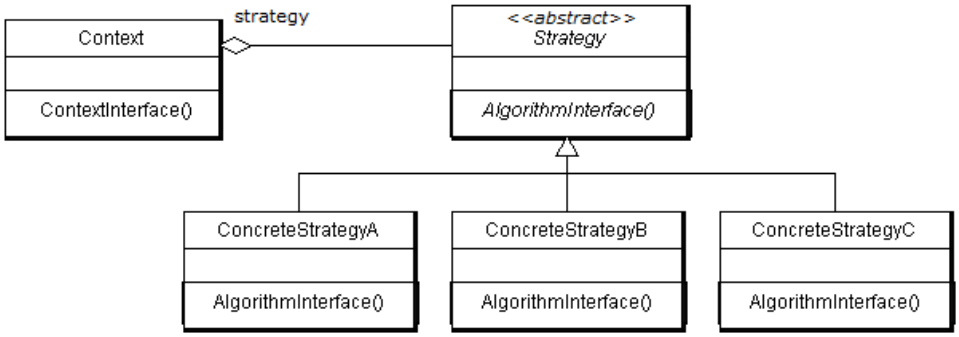


Figure 3. Strategy Pattern

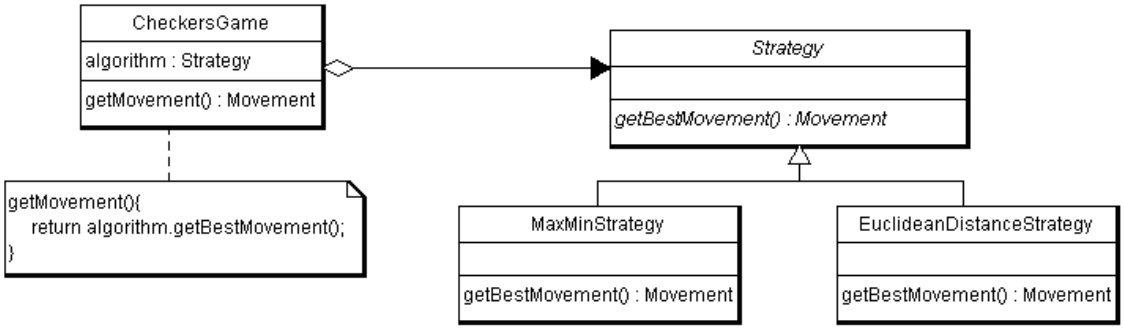


Figure 4. Adaptation of Strategy Pattern