*6th International Conference on Applied Informatics*
*Eger, Hungary, January 27–31, 2004.*

# Teaching technologies at the Institute of Informatics of the University of Debrecen

**István Juhász, Imre Fazekas**

Department of Information Technologies, University of Debrecen
e-mail: pici@inf.unideb.hu, ifazekas@inf.unideb.hu

**Abstract**

Due to the integration of the computer science in our life, computers are used probabily in every office and home. Creators of software have to be up-to-date in new technologies and methodologies in computer science. The main goal is to create IT professionals with ready-to-apply knowledge. The courses introduce the basic concepts of information systems, programming- and metamodels, help to get familiar with design methods, abstract methodologies and so on. IT students acquire both practical and theoretical knowledge. The top of this area is the course named "tehnologies of system developing" contains abstract methods, metamodels, and the practice of the UML, design patterns, RUP and extreme programming.

Besides completing the mandatory classes, students are required to choose section we have. The amount of the classes one most complete from a section is determined by the major of the student. Most curses are manadtory. Thus, every student has to complete this courses on technologies.

**Keywords:** Courses, object oriented, Java, UML, RUP, XP, Design Pattern, Refactoring, Localisation

## 1. Introduction

The following graduate programs is ensured at the Institute of Informatics in the University of Debrecen:

- full-time study: M.Sc. Software Engineering and Mathematics, B.Sc. Software Engineering and Mathematics, M.Sc. Information Technology Teacher
- evening study: B.Sc. Software Engineering and Mathematics
- correspondence study: M.Sc. Information Technology Teacher

The IT-education sector has the responsibility in this field to train highly qualified professionals who can keep up with the progress and learn to use the newest technologies available in order to fulfill the existing and emerging user needs, requirements.

### 1.1. Education

At the University of Debrecen the courses of the IT education are organized according to the following concepts: every graduate program has mandatory classes required for the degree. The knowledge acquired in these fundamental classes serve as a basis for the other classes facilitating the specialization of students. These classes are organized into course sections, where each section except one contains optionally mandatory classes (Section A - Artificial Intelligence, B - Database Systems, C - Networking, D - Computer graphics, E - Image Processing, F - Applied Mathematics, K - Computer Algebra and Cryptography, KTK - Economics). The expression "optionally mandatory" means that students are required to take a minimum amount of courses from each of these course sections. The major of the student in credit points determines the amount of the classes one must complete from a section, but the actual classes taken depend on the students' decision. The courses of IT are the essence of teaching IT professionals and programmers, so it can be found in every sections.

## 2. Courses

### 2.1. Database Systems

The "Database Systems" introduces the basic theoretical concepts of relational and OO databases throughout lectures, while focusing on practical knowledge on relational database systems during the computer labor activities. In the labor activities students study relational database design, normalization and SQL, the standard query language for relational databases by using a concrete relational database management system. We teach all of parts of SQL:

- the Data Definition Language (DDL) to create and drop tables and views,
- the Data Manipulation Language (DML) to insert, delete, update rows stored in database tables,
- the query language (SELECT statement) to query the stored data,
- the Data Control Language (DCL) to grant and revoke user privileges, and to control (commit, rollback) database transactions.

However, during the term the main goal is to drill students in queries. Students become skilled in performing table-joins, using many kinds of clauses and functions in statements dependably.

## 2.2. Programming I.-II.

The students get familiar with the basics concepts and terms of programming, the functional, procedural, logical and object-oriented programming paradigms. The learn how to think using an abstract approach in the programming, how to use the toolboxes of the programming languages like Java, C, Ada, Eiffel, CLOS. In the labor activities students write codes, application fragments, so they learn the theoretical design in practice.

## 2.3. Technologies of Software Localisation

Software Localisation means adaptation of computer software packages and their associated documentation to suit the requirements of different languages. It is more than just changing the language, it also takes into account a number of technical (e.g. text expansion) and cultural (e.g. time/currency formats) factors. This course is a general "sparks" of the process of Software Localisation, and easy-to-use programs will give a better idea of the localisation process during a hole project.

By this course, the student have a closer look at the localisation industry, its past and its future, the roles involved in localisation, the main players and the professional associations. After the course the student will feel familiar with concepts such as internationalisation, globalisation, coded character sets, Unicode, leveraging, QA, simship, etc. and will be ready to enter and explore the intriguing world of localisation. For all team members working on localisation projects, including project managers, translators, terminologists, engineers, it is essential to see the "big picture" of a project. This course aims at providing this big picture. It will not only provide an overview of a typical localisation process, but also produce a real-life case study where a product is taken through all phases from initial publication in a source language to publication in multiple target languages. From source file hand-off to delivery, from project setup to completion, from initial quality check to full proofreading, the course will show the full localisation process and the role of technology in the various process steps.

## 2.4. Technologies of software development

This is a complex mandatory course, containing theoretical and practice-based education. The students study about UML, RUP, XP, Agile development, Design Patterns, Refactoring, and open-source development. Let's audit the training of these technologies.

### 2.4.1. UML, RUP

The UML specification describes a metamodel that defines the elements of each diagram, how the diagrams may be assembled, and how they can be extended. In this cource through two conrete projects, the students learn how to use the UML, how to use development process like RUP, how to improve software quality,

development technologies. The main goal to make the students able to think in abstract way.
The students learn all layers defined by the UML metamodel architecture:

- user object
- model
- metamodel
- metametamodel

layers. To prove a standard development process through the projects, we use the Rational Unified Process.

First of all, we provide the background of the UML and how the students can approach it. We explane what the UML exactly defines, and the mainings of models. They receive an overview of the diagrams supported by the UML and the fundamental object-oriented concepts applied through the development. The learn how to create the Use Case Model, from the diagram through narratives and scenarios to fully document user requirements. The learn to identify and define Use Cases verifyed by a fictive customer. Based on the requirements, they will begin the contruction of the Class diagram, including classes, interfaces, and associations. Applying aggregation, composition they learn the basic restrictions of the structure of a software. The get familiar with the Activity diagram to model logic, such as business processes and complex system behaviors. The start modelling the interactions between objects using the Sequence diagram reusing the information in the diagrams already drawn. Learning the Collaboration diagram, the model object interactions, they can use all unique properties of these tools. For fully understand and document the behavior of the objects in a system, they use Statechart diagrams. The package diagram to teached to organize the components of the software. When, students reaches this point in the design, they need to build the system from the point of view of hardware and hardware components, so they learn the essence of Component and Deployment diagrams.

This course is practice-based, so through the two projects, they implement these simple application to recognize the connection between the diagrams. The RUP is very good tool to coordinate the work of the development. It is essential to understand the importance of guidelines in group of programmers containg more then 50 poeple.

### 2.4.2. Agile methodologies

It is important to ensure a methodology for the small projects too. The RUP is too "big" to handle a development of a small programming group. Agile means a lightweight methodology, that is small, easy to learn and use and doesn't require extensive management support. This project technology treat every person as a real member of the developing team toward the common goal. Main concepts of Agile programming:

- working software
- customer collaboration
- responding to change
- individuals and interactions

### 2.4.3. Extreme programming (XP)

Extreme programming comes basicly from the work of Kent Beck and Ward Cunningham. The main tenets of XP is to include the Change. The traditional development process starts with full and deep analysis and design at the beginning, and hopes to get it right. It is not too hard to recognize that it costs far more to fix some fault in the design after the software is mostly written than to discover and fix the fault early in the design process. The problem of the design is that the requirements always change as the project progresses. For example: understanding of the customer and the developer of the problem increases, making things obvious that weren't obvious at the beginning.

The basic principle of XP is that, software requirements will change, so let's calculate with that change. Use a development process that makes it easy to change software and reduces the cost of doing so.
The students learn the basic practices of XP:

- communication,
- simplicity,
- feedback,
- courage.

Becouse of the rapid rise of interest in XP in the OO programming community, it is essential to teach this methodology to make students become IT professionals. Studens get familiar with the following XP practices:

- the planning game
- short releases
- metaphor
- simple design
- testing
- refactoring
- pair programming
- collective ownership
- continuous integration
- 40-Hour week
- on-site customer
- coding standards

### 2.4.4. Design patterns

Design patterns have become an essential part of object-oriented design and programming. They provide elegant and maintainable solutions to commonly encountered programming problems. These days it is difficult to find an article or book about designing software without the mention design pattern. An understanding of design patterns has become a critical part of any object-oriented programmer's toolbox. The big challenge in developing software is to recognize the problems of developing process int the future. Libraries and frameworks help the programmer to reuse useful tools for many software systems. The tool provided by libraries and frameworks slove common but low-level problems.

Design patterns works at a higher abstraction level, its deal with solving common design problems. They do not provide code that can be used for one problem. Rather, patters help to generate a design solution for a software problem that has come up repeatedly over many software projects. A design pattern means a general description of a problematic situation and a general solution handling the rising fault, so recognizing the situation through a concrete case, its can be adapted and applied to a specific situation. Using design patterns takes some experience.

The students must have at least a basic understanding of all the patterns available, and then be able to recognize when they have a design solution that can be helped by using a pattern. Often, this requires nothing more than reading through the patterns to see if any fit the case at hand. When a patterns fits, it tends to ring a little bell of recognition. The longer you work with patterns, the easier they become to use.

The students got familiar with the basic design patterns, and the practice of design patterns. Through a concrete project, they understand the essence of this design technology.

### 2.4.5. Refactoring

The fact is, most programming involves maintaining and modifying existing code. Refactoring is one of the most recent object-oriented techniques to be formalized and developed into an essential programming tool.

Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure. In other words, it is a disciplined appoach to improving the design of existing code. It should be considered a basic principle of programming and does not require any special methodology. It most powerfull tool to clean up code that minimizes the changes of introducing bugs and failures. In this course the students learn the essence of refactoring, so the method when they refactor they are improving the design of the code after it has been written. With refactoring they can take a "ugly" design, and rework it into well-designed code, and probabely find the balance of work changes.

The students learn from building the system how tp improve design. The resulting interaction leads to a program which design that says good as development

continues. Along the way the students see both the process of refactoring and the application. First of all, we teach the basics or the general principles of refactoring, including definitions, and the reason for doing refactoring. The next move is to describe how to find bad smells in code and how to clean them up. In our course, the testing plays a very important role, so describes how to build tests into code with a simple open-source. The students receive a catalog of refactorings. It's not a full, comprehensive catalog, but containing the most famous refactorings.

We use examples in Java. Refactoring can, of course, be done with other languages. We hope that the essence of this course will be applicable to other languages too. The student got familiar with the basic refactoring process, making attention to reduce risk of change, not to change functionalities, solving only one thing at a Time, and testing each step.

#### 2.4.6. Open source developping

What if the source code of any program is available for anyone to see, use, and adapt? Can it works? Well, open source is a solution of system development, that is usually released under one of several licenses. Even though the source code is open, people can still gain financially by adding value to the software through support and training. The most important example of a successfull open source project is the Linux operating system. In one hand, open source is one way to release and license software, in other hand it can be considered a development methodology.

A set of conventions and customs describing how to use this technology, and how to develop using open source since the beginning of the development process.

## 3. Observations

For the students, the most troublesome part of the courses is the abstract thinking. To design for the future, the create a software structure for the requirements of the morning, to recognize a general case from a concrete ones. Creating IT professionals, the knowledge of abstract design is essential, including the most-recent technologies, and to learn and use the point of view of abstraction is the most difficult problem.

## References

[1] O.- J. Dahl, E. W. Dijkstra, C. a. Hoare. Structured programming. 1978.

[2] Dr Job's Journal, www.dii.com

[3] Object Oriented Programming www.oop.com

[4] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. Pattern-Oriented Software Architecture - A System of Patterns. Wiley and Sons, 1996.

[5] Automatic code generation from design patterns

[6] P. Coad. Object-Oriented Patterns. Communications of the ACM, V 35 N 9, Sept 1992, pp. 152-159.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Reading, MA: Addison-Wesley, 1995.

[8] Schmidt, Douglas. Using Design Patterns to Develop Reusable Object-Oriented Communication Software, CACM, (Special Issue on Object-Oriented Experiences, Mohamed Fayad and W.T. Tsai Eds.), 38,10, October 1995.

[9] http://java.sun.com/docs/books/jls/second_edition/html/j.title.doc.html

[10] http://java.sun.com/docs/books/vmspec/

[11] Robert W. Sebesta, Concepts of Programming Languages, 6/E, 2004 Addison-Wesley

[12] Jim Highsmith, Agile Project Management: Creating Innovative Products, 2004 Addison Wesley

[13] William C. Wake, Refactoring Workbook, 2004 Addison Wesley

[14] Ramez Elmasri, Shamkant B. Navathe; Fundamentals of Database Systems, 4/E, 2004 Addison-Wesley