The Fifth International Conference on e-Learning (eLearning-2014), 22-23 September 2014, Belgrade, Serbia

TRENDS AND WAYS OF DEVELOPMENTS IN SOFTWARE METHODOLOGIES AND PROGRAMMING LANGUAGES

MR TATJANA DAVIDOV
University of Novi Sad, Faculty of Economics in Subotica, tanja.davidov1@gmail.com

***Abstract**: Programming languages are developed from mechanical languages to the languages of higher level. Languages which are orientated towards the object give out the totality of the software task, everything is contained in the object which can support one iteration in developing a certain software solution. The agile approach mobilizes flexible software into a unique developing project together with the frequent cooperation of the members of the project team and clients in order to solve the problems of continually opened tasks of developing new software projects. The agile development is supported by means of component development, upgrading it as partial software solutions which rise the quality of the software task as well as the possible options of their solving and the productiveness of the IT project to be accomplished faster and cheaper. Extreme Programming (XP), as a method of `agile development of software`, implies, among the other facts, programming in pairs where all the members have a joint ownership of the code and frequent direct communication with frequent direct communication with clients at the concrete given spot, location of development.*

***Key words**: extreme programming, XP, object, IT projects, objectively orientated programming, agile development of software, components.*

1. INTRODUCTION

Programming languages are a set of rules, instructions which have a simple, single meaning meant to be given to computers for solving certain programming tasks. They are recognized directly by the computer hardware and can be divided into three main groups:

- machine language, a simple programming language which is internally used by the computer,
- symbolic language and
- high level language.

Machine and symbolic languages depend on computers. High-level languages are mechanically independent languages which use programming instructions in natural, plain and simple words of English language, whereas the operative system and the architecture of the sole computer remains as a completely independent category.

According to the way of solving problems, programming languages are divided into two categories and they are the procedural and the objectively oriented ones (OO). The *procedural languages* which have a programming code, assign the computer a set of instructions which may solve the problem and the algorithm, serves as a means of describing **how** to solve the given computing problem. This is the mode of functioning of all known high-level programming languages such as: Fortran, Cobol, Basic, Pascal, C as well as machine dependent languages. Developing and improving programming languages have been implementing new methodologies such as advancing interpreter that independently by its built-in procedures, steps, solves the described problem. This is the feature of the declarative or non-procedural language for the purpose of writing inquiries (query) as it is the case with the natural language SQL. The objectively oriented languages (OO languages) represent a special class where we can observe procedural and non-procedural elements. In procedural languages the actions, ways of realizing procedures and tasks are emphasize where the action proccesses the data by performing the action onto the given data. By the OO approach the programmer tries to wrap, put the data and actions into the tool box (framework) which has the main role in speeding up the development of the software. Small Talk, Java and C# are the best examples of OO languages. The great manufacturers of software today have their own programming packages such as Microsoft`s® [1] set of applications named Visual Studio Net with C#, currently the most popular tool for fast development of an application (Rapid Application Development - RAD). This developmental surrounding contains a series of redefined components, forms and wizards which all together makes strong support to the development of software through all of its phases. The new developing surroundings and OO languages lessen the robustness of the software. They divide the software into partial moduls while the development of the software solutions is based on the objects and interactive units which contain complete functionality for the given programming task implemented in themselves. The defined programming task is repeated several times from one system into the other which forms the mission of standardization of independent software solutions. The development of the ultimate Internet created the utmost need for the written projects to operate in the same way, identicl in different platforms (operative systems). When the OO programming language Java was developed, programmers found solutions and various possibilities of overcoming problems connected with executing software applications

[1] Microsoft is the protected brand of the Company

in numerous and different platforms. The comlex development of software solutions was met with all sorts of thinking, approaches in upgrading and its realizations, ways of programming and methods of making software, the architecture of the system. Historiclly looking, it evolved from primitive to highly sofisticated and modern developing techniques of creating software solutions as it follows:

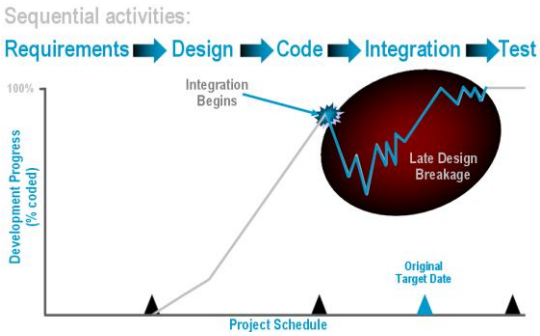
- writing programming codes - solving problems by programming them until 1970 is also present nowadays,
- structural methods of developing software: analysis and projecting until 1985,
- development of software based on the models of data, data base and the languages of the fourth generation, since 1980,
- objectively oriented methods of development, since 1980 and
- standards - the UML model of segmental or interactive development of software, since 1998.

2. WAYS OF DEVELOPING SOFTWARE

The traditional model of developing software was popularized as a Waterfall Model. The origin of its name is connected with the article of W. W. Royce which was published in 1970. Royce presented the model of a waterfall claiming that even such a rigid phase system may be developed into a flexible interactive developing process.

It supports all six phases which are developed in the most stable way and undisturbed follow each other, without skipping or shortening the phase tasks: analyses of demands, design projecting, implementation, testing, integration and maintenance. This model of developing software has been used for years and is still applied for its numerous advantages which are not to be neglected:

- better spent time in earlier phases may lead to economical and more efficient work later,
- putting accent onto documentation and authentic code of software products,
- makes progress linearly through discrete and easily understood defined phases since it is rather simple,
- spots control places during the developing process.



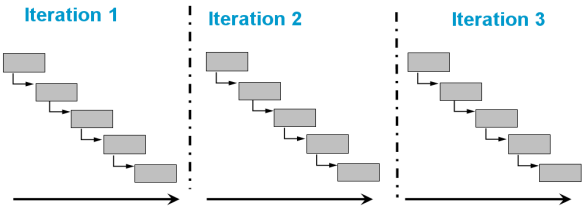
Picture no. 1: Practical implementation of a Waterfall model (literatura [5])

The experience of software teams, project and software solutions tell us that such projects must be open and ready to accept changes, taking into consideration the fact that clients tend to change their demands. Regarding the fact that OO programming promotes an object from the outer world, as an independent programming unit, where objects are shown and organized in a systematic way making them logically understood and adaptable to potential changes, specific demands, while supporting the analyses of the leveled system. Thus, this model contributes to the following:

- precise formation of the complete future structure of the informatics, computing system,
- formal and logical projecting of the work frame of the future system,
- creating operating tasks by stages according to the objects and the project decisions,
- making the prototype of a new system,
- realistic and versatile evaluation of project decisions in order to choose optimal (economic and productive) solutions and,
- creating files of all the steps during the realization of the new system.

The OO approach to the development of the software solutions bonds the data and the processes into a single logical unit which is defined as a class. In that way a part of the software is isolated from the rest of the system because the changes inside the class have no impact onto the remaining parts of the system, classes in the system itself. The independence of the class provides it with the opportunity to be used in very many ways over and over again in different applied platforms. In OO languages the software is developed by the objective approach and ways of thinking and that means the following connotations:

- there are no classical functions and procedures, just objects,
- tasks are worked out in several smaller, less developed separate steps,
- the work out goes from the top towards downside,
- data have a smaller role, impact compared to the functions,
- fundament, the base of the initial development is made of objects, not by functions and procedures,
- the object is an abstract element in the field of its usage and application.



Picture no. 2: Iterative approach in developing software solutions (Literatura [5])

This approach implies the iterative development. The life cycle of the software contains iterations where each iteration has an independent, separate software solution according to the standard phases of its development: analysis, design, programming and testing. The first few phases promptly discover the risks and failures, and each

iteration is developed swiftly and fast from the beginning to its end. In the course of its development each phase is tested several times and integration of other solutions are also possible in numerous software projects.

Different approaches in projecting software systems that promote the leading experts such as: Jacobson, Booch and Rumbaugh unite themselves into one and unique process (Unified Process - UP). As the sponsor of this methodology was the company Rational (well-known software firm which deals with the development of CASE tools) it is not surprising that this process was named Rational Unified Process (RUP) and belongs to the group of easy agile processes. The collaboration of the above mentioned authors resulted in a unified modelling language (Unified Modelling Language - UML) onto which the RUP leans to. UML was developed in 1997 as an approved standard by the Object Management Group – OMG (Literatura [6]). UML is a standard language for specifications, projects, visualizations, constructions and recording projects of the software systems which are developed by the means of OO technology. It is extremely flexible and may be widely used in creating working, operative tasks and it is not necessary for it to serve to the objective oriented technology.

The strategic values of a software are rising in lots of companies, the industry seeks for the techniques which will transform the production of software into an automatic process which will improve the quality of the products and lessen the costs of the final product and help it arrive onto the market faster. Such techniques are developed by using component technologies, visual programming, forms and frameworks. Companies are constantly trying to increase their competences on the market. The solutions which are of special interest are those which refer to: physical distribution, competitiveness, data replication, security, balancing of the overloaded system and the tolerance of errors. UML is designed to answer onto these and similar demands. UML is a technique globally exploited for two main but different developing ways, from the visual software project solution to the programming code, in other words it may be claimed that it forms the initial visual project from the programming code. Projects have a detailed graphic description of the system and it serves as a basis for writing the programming code, while some of them even generate parts of the code. In the prototype developing of the software CASE tools are used for the realization of initial visual project solutions. UML generalizes programming of project solutions in several steps:

- visualization of the system, presenting its outlooks,
- specification of the system, its structure and behavior,
- files and records of the project decisions.

3. NEW DEVELOPMENT METHODOLOGIES

The methodology for developing a software is a complex sum, number of procedures which provide advantages during the development:

- consistent approach lessens the risk of errors,
- complete documentation for the current and future projects is written down,
- quality of the software is regarded high if the solution is agile and susceptible to changes,
- changes of the project teams are also a common practice, experience and knowledge are decisive in choosing the best members,
- visual modelling is easier and so is adapting compared to searching for programming code or changing it.

Nowadays, the agile process which is more and more present is the adaptive one. It was invented in Utah, USA, in 2001. The most important agile methods are: Scrum, XP, Open UP, Crystal, Lean, DSDM and Agile MSF. This process of development is more adaptive, agile and the cooperative work of the team members working on developing software solutions becomes more effective. Agile methods are based onto four main principles:

- people and their relationships are much more important than the process and the tools,
- software functioning comes before recording its documentations, which is not of the greatest importance,
- cooperation with the final consumers is more important than immediate negotiations of working tasks,
- response onto necessary changes and computing tasks are far more important than planning steps and activities.

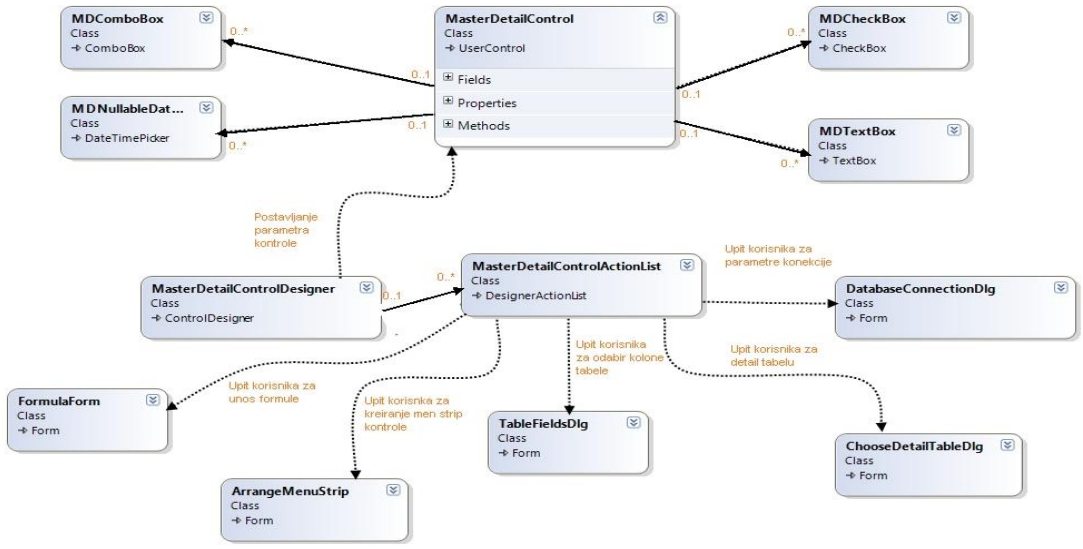
Generators of application prototype serve for generating the authentic code in OO language. Such a prototype represents a client`s application which by the means of the protocol for the base connection (Open Database Connectivity - ODBC) communicates with the server for the distance, relation database. These tools are especially suitable for a fast development of the system, since they create the user`s interface, state precisely the outlooks of the screen and the format of the user`s reports. Since the class is the main term of the objective development and projecting, class diagrams are mostly used in describing the structure system which is being developed. Characteristics and operations are terms used for describing the class features. UML, as a complex language, contains a great number of diagrams, which in a certain phase of modelling are used. They are also strongly connected with the process of creating a software. The generator of the programming code has a number of generators for various different purposes. SQL generator produces a project and implements its outline for the database. In accordance with the standard syntax, nonprocedural SQL language generates the programming code which contains all the commands for executing the following programming operations:

- creating tables,

- declaring limitations in the form of primary and strange keys,
- declaring limitations and permissions which are in relation to the value of the zero traits,
- declaring limitations in the form of the referential integrity, in the case of its violation, generating the mechanism to returning the database into its consistent state,
- creating triggers and procedures for controlling limitations in relations, if the situation allows the usage of the SQL language in the given problematic case.

The previously described generators of the application prototypes served as an example for software possibilities and they led us to the conclusion that it is extremely productive to develop a generative component which is able to generate the demands of the system, the database and the final users. In the domain of the Visual studio, by the objective approach, the software component was developed for generating applications of which the diagram class is described in few spots of the iterative development.

The main, entrance spot is in the component of the MasterDetailControl. In order to make the memorandum (master/detail) control to generate all of its fields, the parameters for connecting the database is its priority, and they are the name of the server and the database. These parameters are sufficient if the Windows authenticity is used. By the means of these parameters, the control board finds the tables of the database and offers the possibility to choose options out of the Master table. From the information scheme we get data on the chosen table such as the names and types of the columns, primary and strange keys, tables, details, etc. The user chooses this data on the bases of defined terms, the values of the fields are generated as well as the labels, connections among the tables, visual and functional description of the fields and such options in connection with the fine adjusting of the way how the model looks like. When it comes to the implementation, everything is in the class of MasterDetailControl. The class DataSet exists as an attribute in the MasterDetail Class and it reads the tables, attributes and all the rest of date from the tables. Everything the user does has an impact onto the data stored and saved in the DataSet. Only after clicking onto the DataSet, it synchronizes the data with the previously saved data. The fields, forms, master and detailed tables are all connected in the corresponding columns in the DataSet and by their editing they are written down, copied into the DataSet automatically. The diagram class MasterDetailControl – the component which generates the cipher reader and other documents such as the memorandum/items are shown on the following picture:



Picture no. 3: Class Diagram of MasterDetail Control client/server components

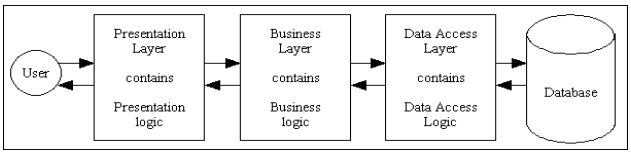
4. NEW METHODOLOGIES CREATE REUSABLE VALUES

A different approach to the development of software by the means of agile software methodologies, such as the Extreme Programming (XP), one of a several fast and agile methodologies, gives us the opportunity to adjust swiftly to the changes in the informational demands, during the process of developing the project. Because of all the above mentioned facts, it is more realistic and accessible than the previous ones which insist on defining, stating all the demands at the beginning of developing the project. The constant interaction between the clients and the members of the developing team leads to a high-quality ending of the work, cutting down the costs, continual management of the demands of the clients and absolute flexibility in developing the system. Extreme Programming is a model of developing a software especially for small and medium-sized developing teams which are constantly exposed to permanent fast and changeable developing activities and demands of the informatics systems. XP offers a great number of formalized solutions which are defined in: programming in pairs, unit testing, refactoring, steady changes according to the new demands, architecture, as well as frequent and repeatedly short iterations which are fully

developed. Such a quick answer onto these rules of the extreme logic in programming is widely supported by the application of the component development of the software, where each component represents a built-in system unit which functions separately but may be also integrated into a complex system of the complete software solution. This solution is safe, agile and adaptive, flexible and may be easily adjusted to the system and clients` demands, reliable and accurate, tested and previously used several times.

Software is goods which does not have amortization. Once it is created, without greater oscillations and on the top-quality hardware platform, may last a long period of time, with the exception of its obsolescence (new operative systems, more perfect, sophisticated and comfortable platforms for developing and maintaining the software, more rational approach to the database, better modern interfaces and reports, etc., in fact, it is relatively questionable and non-objective to discuss what an iternal or obsolete software means). The growth and the development of a software have taken the precedence in the industry of the XXI century by their agile approach to the development and improvement and their only limitations are in the human ambitions. Components, as priorities in developing software systems, are made on a three syllabic architecture. The users` interface is separated from the business logic and the layer of the data. Such a solution enables reusable functions:

- User interface layer – graphical user interface - (GUI),
- Business layer – business logic layer - (BLL):
  - *Process components* – local business functionality,
  - *Business domain components* – functionality of business processes,
  - *Business infrastructure components* – functionality of business domains.
- Technical infrastructure layer – serves to business components - layer which enables access to the data, their physical accommodation onto the server disc of the database (data access layer - DAL).



Picture no. 4: Layers of Component Architecture (Literatura [5])

The component architecture is flexible, contains the complete solution, key components and subsystems, mechanisms of its integration and realization, integrated solution, the communication of its mutual processes and the formulae for executing certain tasks. This architecture performs the following tasks:

- divides tasks in the project team,
- maintenance and spotting the possible ways of expanding, spreading,
- estimates the profitability of the reuse or makes it possible for wider use,
- makes choices among thousands of ActiveX and Java components and,
- increasing evolution of the existing software.

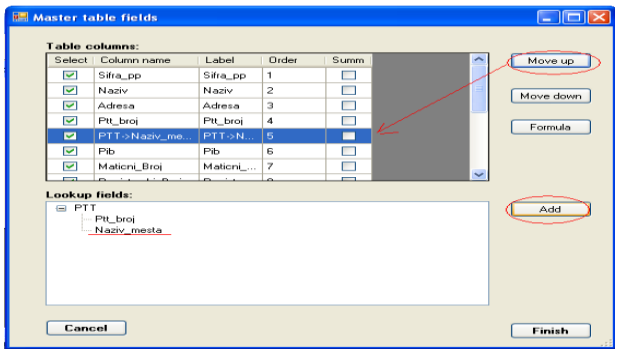
In the applications of the Microsoft Windows developing surroundings, control elements or components are defined as programming units which perform incoming and outgoing operations. As a part of the developing project, controls are in the form of a dialogue or a pattern where they serve to all users, members of the project developing team, programmers and may be used in each application of the framework Dot Net. Most of these components are in the library of standard control boards. The library was created for the need of making the users` surroundings apt and persistent to the standards where the control was defined in advance (Predefined controls) or to fit classes of the framework Dot Net. The methodology of making programming systems based on the components (Component-Based Scalable Logical Architecture - CSLA), by the author Rockford Lhotka [Lhot98], has very detailed instructions in the project domain whereas it is more simple and general in the domain of analyzing the demands. It is well oriented according to the Microsoft technologies, with the aim of multiple reuse of the programming components in different work frames and the users` interfaces (client/server architecture, Web platform). Lhotka components are divided into three categories: components for general purposes (standard components), components created for a specific aim for a single application and components made for a specific purpose for just one type of industry.

The created software components, or as we popularly call them R values, are approved my certain number of international component standards, are added to the library of the software components where their formal list, classification and outlooks, features are described in full detail. When the R values are described in this way, they become a part of the licensed list of the other R values which are already in the library of the software components. The users of the software components, on the basis of the detailed, precise description of the R values, its grades, may understand and estimate if such a R value was suitable for them and their needs. The description of a single R value contains technological and informational elements which are presented, clearly shown to the users in a uniform, previously stated schemes. These values are greatly used in creating systems and software goods (applications), or modifications of the existing ones. On the industrial market they are being sold as any other type of goods which has its usage and cost. According to the given description of its components, the buyer is responsible for choosing one in order to use it in an appropriate way. The users choose these components which are easy to be adjustable to their tasks and problems which have to be solved. Making the right choice is not an easy task because of the individual and creative character in the process of developing the system, its surroundings, differences in which the values are being accepted, its

software and especially since there is not a "formal recipe" for choosing the right component.

Each component must have a simple, clear and lucid design of the total developing environment, the controls in the system and all of these must be easily understood by all the members of the project developing system, whereas the component solutions must show a clear picture of the system to each category of the users.

A generative software component was formed in this research and it has a complete model of the demands and the design which generates software applications. Its usage is equalized according to the needs of the final consumers, the architecture of its development is unique, it is easy to be used by all the members of the project developing team. The component executes all the appropriate tasks, while the users` design and its functionality is completely understood to everyone. The quality is proved by its wide-spread and multiple implementations and direct testing of them, while its standard developing architecture lessens major technical, formal and functional risks from the very start.



Picture no. 5: Generating master form from the database table

The generative components which perform different functions are perspective software solutions and have a major, leading role in developing projects with the aim to form new applications founded on powerful technical and methodic approach which, at the same time, are adaptive, simple and fast in accessing the development of the applicative software understood and accessible to everyone. The high level of the economic justification of component solutions originates from the possibilities of their multiple application. The OO generative component is a software tool which makes a new applicative software. Creating new applications means lots of new tasks and repeated activities on designing a user`s interface, menu, forms, reports and accesses to the database. Thus, the automation of executing all the mentioned and similar operations is a generative tool for producing a brand new applicative software. Its practical application helps the programming teams to develop different domains in creating automatic ways of developing new applications. In that way business tasks are solved quickly and easily while saving money, time, materials and human resources.

5 CONCLUSION

Agile methods of developing software put a great emphasis onto the communication among the people who make projects, programmers and final consumers, clients. They solve concrete defined tasks and give answers onto the current and frequent changes in the field of informatics and computing demands. In that way the newly created software solution becomes completely adjusted to the needs and expectations of its client. The phases in projecting, developing tasks and solutions together with the documents and recordings, are not priority. They are all formally a less important segment of the final, improved product. Agile methods have a lot of positive features, but there are also inevitable problems as in many other methodological approaches. The agile development by strict rules cannot and should not be formalized since it is unpredictable how the clients and members of the project team will behave. The roles of both sides in this case are extremely emphasized. Clients make great effort, the project documentation and files are also great, sometimes it is not fully defined, precise and clear what clients expect. The legal side of the project and its documentation may not be adequate expressed since the agile contracts are without full, legal formalities. So, there is no way to cover all the legal problems on both sides. But one thing is in common and it is **speed**. Time and mutual trust is of great significance. The problem of functional spots and the budget still represent the cornerstone of the future agreement. The agile team is always expected to adjust itself continually which creates new changes in developing the project, and makes it hard to meet the deadlines. If the project and the team responsible for its development are both agile, one question has the utmost importance: what is the real time needed for finishing the project and what is the real cost of the software? The clients themselves have to be agile and flexible especially when it comes to terms of finishing the project, paying the costs and implementing the software. However, the application of the agile methods in developing software greatly contributes in raising the level of quality and the productiveness of the IT projects.

LITERATURE

[1] James Shore and Shane Warden, *Agile Development*, O'Relly, USA, 2007.

[2] Andy Ju An Wang, Kai Qian, *Component-Oriented Programming*, Hardcover,USA, 2005

[3] John Cheesman John Daniels, *UML Components: A Simple Process for Specifying Component-Based Software*, Paperback, USA, 2000

[4] MSF *for Agile Software Development Process Guidance:* http://www.microsoft.com/downloads/details.aspx?FamilyId=9F3EA426-C2B2-4264-BA0F-35A021D85234&displaylang=en

[5] Prof. Ph.D. Angelina Njeguš, *Information Systems Design*, (Projektovanje informacionih sistema) Singidunum University, Belgrade, 2009/2010.

[6] OMG international UML standard which represents the unified standardization in modelling, "the Unified Modeling Language"™ (UML®), http://www.omg.org/