Software Development: Agile vs. Traditional

Marian STOICA, Marinela MIRCEA, Bogdan GHILIC-MICU
Bucharest University of Economic Studies, Romania
marians@ase.ro, mmircea@ase.ro, ghilic@ase.ro

Organizations face the need to adapt themselves to a complex business environment, in continuous change and transformation. Under these circumstances, organization agility is a key element in gaining strategic advantages and market success. Achieving and maintaining agility requires agile architectures, techniques, methods and tools, able to react in real time to change requirements. This paper proposes an incursion in the software development, from traditional to agile.

Keywords: *Software Development, Traditional Models, Agile Models, Agile Architectures, Agile Techniques, Agile Instruments*

1 Introduction

Increased agility is a magnet to all organizations, especially for those in private sector. It will allow a rapid and efficient adaptation to market changes and gaining a strategic advantage. Moreover, increased agility contributes to decreasing the development time for new processes and increasing flexibility for existing processes, where modification and implementation is required. All that leads to decreased time for solving client demands, more clients gained, lower adaptation costs and finally increased revenue.

In a complex and permanently changing environment, organization agility is no longer a necessity but a condition to access or remain on the market. An agile enterprise adapts fast to client demands and market opportunities, gaining competitive advantages on the market. This can be achieved only if the stakeholders clearly understand the working ways of the organization.

From informational systems point of view, there are several ways to achieve agility, among which: Business Process Management (BPM) for orchestrating independent functionalities, Service Oriented Architecture (SOA) for designing and developing such functionalities and Decision Management (DM) for management of organization decisions.

2 Software Development Life Cycle

Development models are various processes or methodologies, selected to develop the project according to its purpose and objectives. Software developments models help improve the software quality as well as the development process in general.

There are several models for the software development life-cycle, each developed for certain objectives. Software Development Life Cycle (SDLC) is an environment that describes activities performed in each stage of the software development process. SDLC consists of a detailed plan that describes how the development, maintenance and replacement of specific software is conducted. This is also known as software development process. [1]

The international standard for SDLC is ISO/IEC 12207. It aims to define all activities required to develop and maintain software. Figure 1 depicts the various stages of a typical SDLC.

DOI: 10.12948/issn14531305/17.4.2013.06

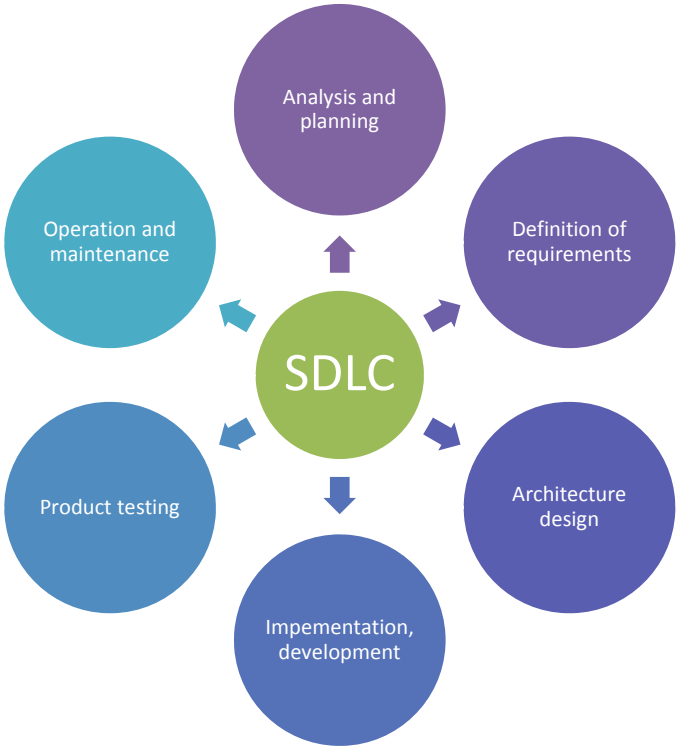

Fig. 1. Software development life cycle [2]

Stage 1: Requirements analysis and planning
Analysis of requirements is the most important stage in SDLC. It is performed by senior members of the team, using inputs from the clients, sales department, market research and industry experts. This information is then used for a basic project plan and feasibility study from economic, operational and technical points of view. Also, in this stage the team plans the quality insurance requirements and identifies project risks. The result of technical feasibility study consists of definition of carious technical approaches that can be used to implement the project with minimal risks.

Stage 2: Definition of requirements
After the requirements are analyzed, product requirements are clearly defined and documented. They must be approved by the client or by the market analysts through SRS (Software Requirement Specification). The SRS document lists all product requirements that must be designed and developed throughout the project life-cycle.

Stage 3: Product architecture design
SRS is the basic reference from which the architects set out to create the best architecture for the product. Usually, at least one product architecture approach is proposed, and it is documented in a DDS (Design Document Specification). This DDS is revised by all interested parties and the best approach is selected, based on some parameters like: risk evaluation, product robustness, design method, budget and time constraints.

A design approach clearly defines all architectural modules of the product, along with communication and data flows to and from external modules provided by third parties (if there are any). Internal design of all modules in the proposed architecture must be presented in clear details by the DDS.

Stage 4: Product implementation or development
In this stage of the SDLC the product development starts. The source code is generated during this stage. If the design was performed in a detailed and organized manner, the source code can be performed without complications. Developers must follow the guidelines of their organization. In order to generate the code they use programming tools like compilers,

DOI: 10.12948/issn14531305/17.4.2013.06

interpreters, debuggers etc. The source code is written in high level languages like C/C++, Delphi, Java, PHP. The programming language is chosen according to the software being developed.

Stage 5: Product testing

This stage is usually a subset of all the stages in modern SDLC models, because testing is involved in all SDLC stages. Still, this stage only involves the situation where product faults are reported, tracked, fixed and re-analyzed until it complies with the SRS quality requirements.

Stage 6: Market operation and maintenance

Once the product has been tested, it is ready to launch on the market. It can be launched on a limited segment and tested in a real business environment, then, based on feed-back received, it can be launched to the whole market unchanged or with improvements suggested by clients involved in tests. After the launch, the maintenance is performed for the existing client pool.

3 Software Development Models

There are many software development models and many organizations create and use their own model. Choosing the model has a high impact on testing. The independent phases, applied on all levels are: testing and validation; management. Among the most widely development models are:

- Waterfall model;
- V model;
- Incremental model;
- RAD model (Rapid Application Development);
- Agile model;
- Iterative model;
- Spiral model.

Each model has advantages and drawbacks and must be selected according to organization needs. For space reasons, the following sections will present a brief description of stages, advantages and drawbacks and usage [3], [4], [5] for only two of these models: waterfall model and incremental model (this one being the base for all agile software development models).

3.1 Waterfall Model

The waterfall model was defined by Winston W. Royce in 1970. It is also known as linear-sequential life cycle model. This model is easy to understand and use. Each stage must be completed before next one can start. At the end of each stage the project is reviewed to ensure compliance with requirements.

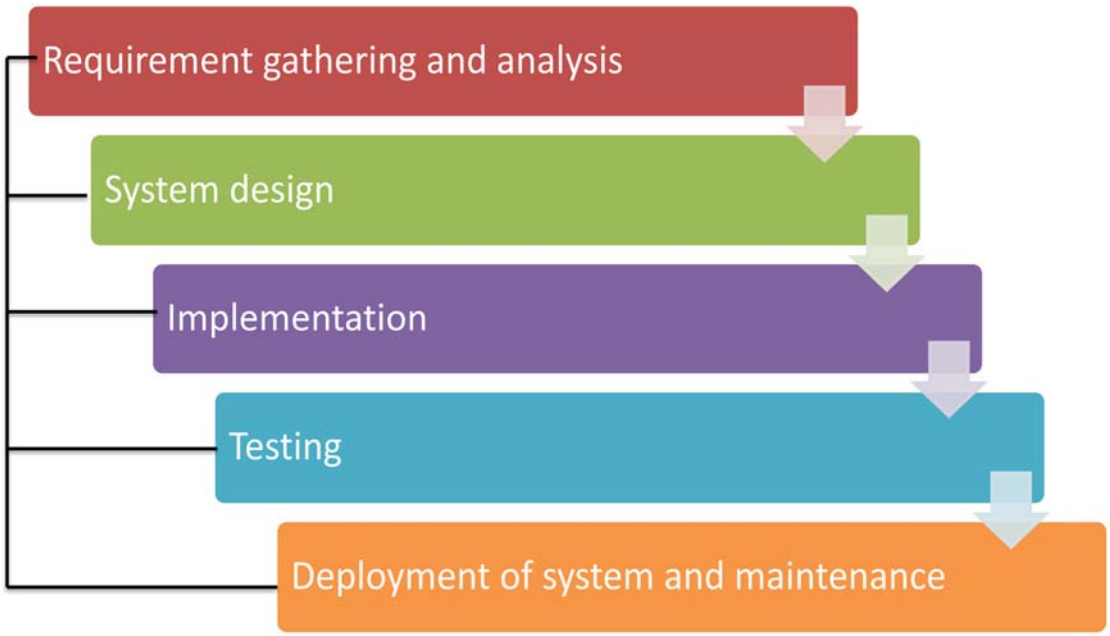

Fig. 2. Waterfall model diagram

DOI: 10.12948/issn14531305/17.4.2013.06

Some of the **advantages** of this model are:

- the documentation and structure design are an advantage when new members join the team;
- it is easy to understand and use;
- it is easy to coordinate due to the model rigidity – each stage has an expected result and an evaluation process;
- stages are implemented one at a time, in sequence;
- it is recommended for small projects, with requirements clearly understood.

Some of the **drawbacks** of this model are:

- some requirements may arise after the initial requirement gathering was completed, which influences negatively the product development;
- not all problems detected during a stage are completely solved during the same stage;
- there is no flexibility in partitioning the project into stages;
- new requirements added by the client lead to additional costs, because they cannot be solved in the current edition of the product;
- it is difficult to estimate the time and budget for each stage;
- there are no prototypes until the life cycle is finished;
- if testing detects some problems, it is very difficult to return to design stage;
- there is a high risk and uncertainty;
- it is not recommended for complex and object oriented projects.

The **waterfall** model is recommended for the following cases:

- requirements are well understood, clear and final;
- product definition is stable;
- technology is understood;
- there are no ambiguous requirements;
- resources that involve expertise are freely available;
- it is a short project.

3.2 Incremental Model

In the incremental model the requirements are divided into subsets. The model involves multiple development cycles, which makes the life cycle look like a "multiple waterfall" model. The cycles are again divided into smaller cycles, modules easier to manage. Each module goes through requirement analysis, design, implementation and testing. During the first module, a working version of the software is created. Each following version adds new features and functionalities to the previous one. The process continues until the system is completed (Figure 3).

Some of the **advantages** of this model are:

- each stage delivers a working product, that meets some of the client requirements;
- prototypes are delivered to the client;
- client feed-back is distributed throughout the entire development process;
- it is more flexible – involves lower costs when purpose and requirements change;
- it is easy to test and debug during a small iteration;
- cuts down on initial delivery costs;
- the risk is easier to manage because all risks are identified and managed during the iteration;
- when there are new requirements, they can be introduces in the next prototype.

The model carries some **drawbacks** too:

- it requires a good planning and design;
- requires a clear and complete definition of the entire system before it can be divided and incrementally built;
- total cost is higher than the waterfall model;
- design errors are harder to fix and remove;
- incremental approach may easily turn into "code and repair".
- the client can see what can be done and can ask for more;
- object oriented approach provides a comfortable framework for evolution development, in an iterative manner.

DOI: 10.12948/issn14531305/17.4.2013.06

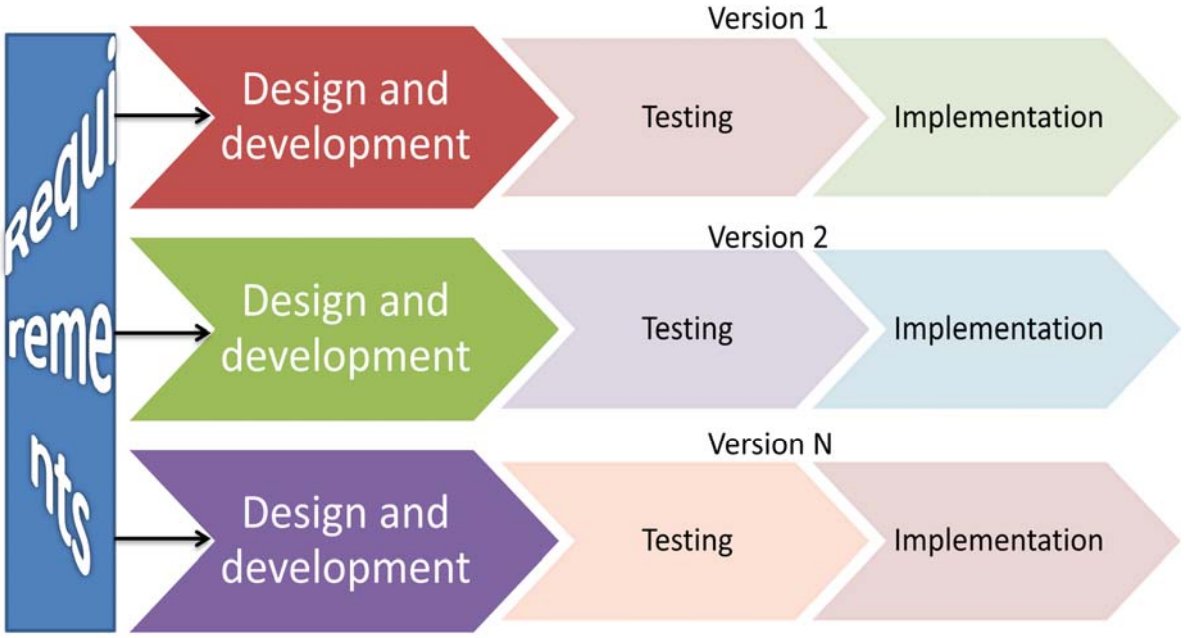

Fig. 3. Incremental model diagram

The incremental model is recommended for the following cases:

- system requirements are clearly defined and understood;
- major requirements are final. Some details may change in time;
- early launch is required;
- a new technology is used;
- there are high risk characteristics and objectives.

4 Gaining Business Agility

The key element of business agility is the Service-Oriented Integration (SOI) and the guaranteed path to SOI is SOA. [6] SOA is used by organizations to increase their agility and flexibility for dynamic adaptation to the business environment. BPM has proved its efficiency in reconfiguring processes to gain agility. DM helps automate decisions and allows business policies to be moved to a central repository. This leads to better substantiated decisions and furthermore to increased organization ability to answer to market changes and opportunities. Achieving agility requires knowledge of how SOA, BPM and DM can lead to increased organization agility and innovation.

To become agile, the organization must permanently control the dynamic of its own business processes, human resources and informational system. Business agility can be achieved through: human resources agility, business processes agility and information technology agility. Service orientation may be applied to organization level with beneficial effects on human resources, business management and information technology (Figure 4).

Human resource agility can be achieved through an organizational culture that provides intensive business and technology knowledge based work force in all the areas of the business. Organizational culture must allow a quick adaptation of the work force and autonomy of work groups. These allow global adaptation to micro changes in business environment.

Business process agility can be achieved through business process management, based on organization informational systems. The principle that lays at eh foundation of agile organization regarding business processes is providing the best solution to achieve the organization purpose. This involves continuous modeling, simplification and reconfiguration of processes. Also, it

DOI: 10.12948/issn14531305/17.4.2013.06

involves the ability to efficiently answer to changes in the business environment by using the advantages of IT services and SOA. Furthermore, combined use of BPM and SOA facilitates a new stage of flexible business process development, known as service oriented business processes.

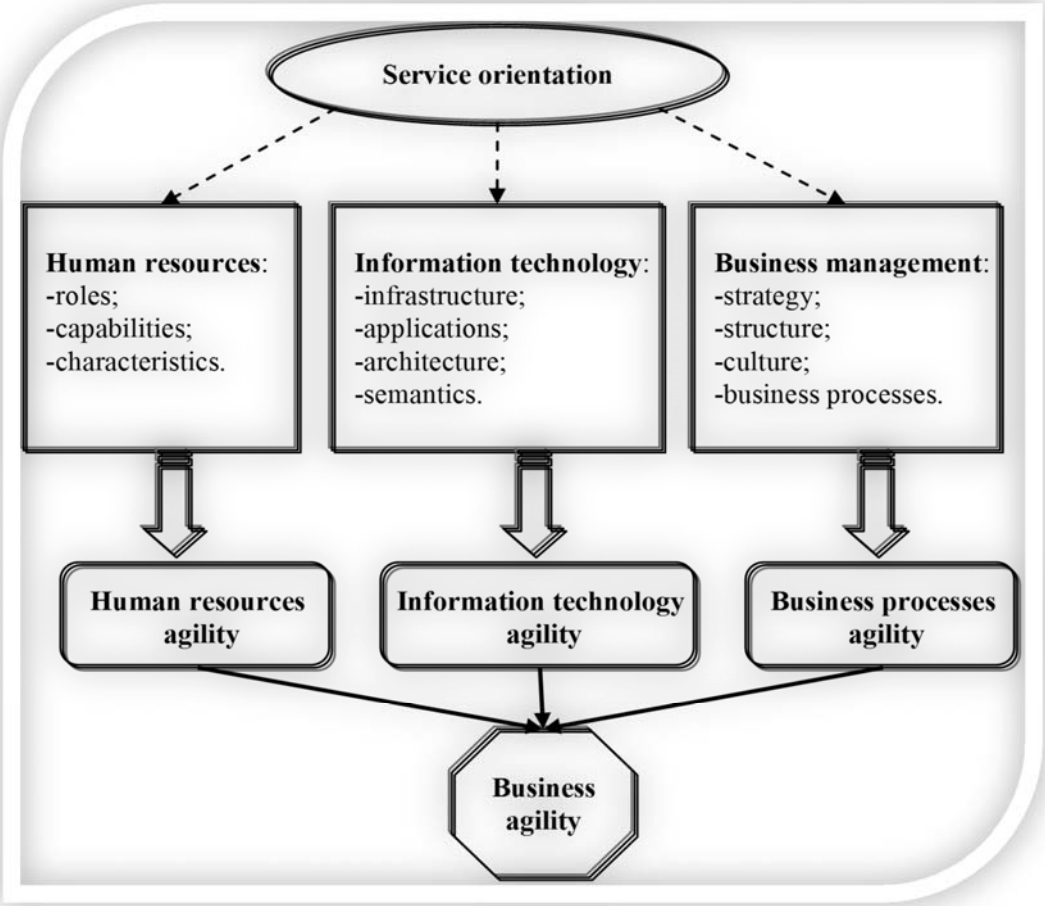

Fig. 4. Service orientation influence

BPM helps increase the agility of the business by providing process models easily adapted to the internal and external business environment requirements. Some of the instruments that may be used to achieve agility of business processes are: Business Process Model and Notation Designer, Business Process Execution Language Server, AJAX Integrated Development Environment, Enterprise Content Management.

From the business management point of view, in a complex decision making process the managers are faced with difficulties in making fast decisions with low costs. DM can be used to help automating decisions, especially the operational ones. DM helps raise the quality, consistency and agility of decisions, in the same time lowering the time and costs associated with the decision making process. Also, decision management, as well as process management, leads to simpler, more agile and effective client oriented processes. According to [7], the DM solution may be defined in four areas: analysis and collaborative planning, knowledge acquisition and learning, unified access and analysis of structural data and unstructured content and automation of intelligent processes.

SOA allows DM technologies to implement business policies as decisional services that

DOI: 10.12948/issn14531305/17.4.2013.06

deliver functionality. Decision services contain all the conditions and actions required to make an operational decision. Each decision service responds to a business questions for the other services. Combining DM capabilities with BPM helps fast implementation of changes in answer to market pressure, reusing the decision logic in processes and systems, making the best decision based on a list of decisions, optimization of business processes, integration of intelligence in business process and gaining competitive advantages. Also, using DM helps adaptation to changes and aligning IT systems with business requirements.

Information technology agility addresses mainly the technological architecture and infrastructure through SOA. Technological agility currently represents a wide field of research, because IT creates most problems in achieving agility. Moreover, integration into organization represents the least mature category of capabilities, for many organizations being a goal and not a reality. SOA may be a paradigm that can solve integration problems on application level.

SOA provides agility through embedding application logic into services. Services can be fast and easy combined with business rules and analytical services in order to provide new functionalities and business agility. The principle that lays at the base of the organization regarding information technology is systematic anticipation of rational use of emerging technologies.

Achieving agility begins with removal of barriers and, usually, the informatics system has most of them. In order to respond to continuous change requirements, businesses must integrate and connect various systems and data sources, many locked into isolation or into proprietary systems.

5 Agile vs. Traditional

Agile methods are based on adaptive software development methods, while traditional SDLC models (waterfall model, for example) are based on a predictive approach. In traditional SDLC models, teams work with a detailed plan and have a full list of characteristics and tasks that must be completed in the next few months or the entire life cycle of the product. Predictive methods completely depend on the requirement analysis and careful planning at the beginning of the cycle. Any change that is to be included will go through a strict change control management and prioritization. The agile model uses an adaptive approach where there is no detailed planning and only clear future tasks are those related to the characteristics that must be developed. The team adapts to dynamic changes in the product requirements. The product is frequently tested, minimizing the risk of major faults in the future. Interaction with the clients is the strong point of agile methodology and open communication and minimal documentation are typical characteristics of the agile development environment. Teams collaborate closely and often are located in the same geographical space.

While agile SDLC is better suited for small and medium projects, on large scale traditional SDLC is still the better choice. Therefore it is important that the development team selects a SDLC that is best suited for project at hand. There are criteria that can be used by the development team to identify the dimension of the desired SDLC. They include team size, geographical location, size and complexity of the software, project type, business strategy, engineering capabilities etc. Also, it is very important for the team to study the differences, advantages and drawback of each SDLC before making a decision. Furthermore, the team must study the context of the business, industry requirements and business strategy before evaluating the candidate SDLCs. It is important to have a SDLC evaluation and selection process because it maximizes the chances to create successful software. Therefore, selection and adoption of an appropriate SDLC is a management decision with long term implications.

Although agile methodologies triumph over traditional ones in several aspects, there are

DOI: 10.12948/issn14531305/17.4.2013.06

many difficulties in making them work. One of them is the significant reduction of documentation and the claim that the source code itself should be the documentation. [8] Thus, developers used to agile methods tend to insert more comments in source code in order to clarify and explain. It is difficult for beginner developers or new members of the team to complete their tasks when they cannot fully understand the project. They ask lots of questions to the experienced developers and this may delay completion of the iteration, which can lead to increased development costs.

On the other hand, traditional methods emphasize documentation in orientation and clarification of the project for the development team, so there is no concern about not knowing the project details or not having a knowledgeable developer.

Agile methodologies are well known for the importance given to communication and client implication. [9]

For each version delivered, the development team and the clients will organize a meeting where the team will present the work done in current iteration and the clients will provide feed-back on the delivered software (improvements on current features or addition of new ones).

Most times, developers will find the periodic meeting (usually weekly) boring and tiring because they have to present the modules repeatedly, to new members and clients and, on each iteration, changes may happen as requested by clients.

The time frame for each iteration is short (usually weeks). Developers will find the schedule too tight for each module, even more so for modules that require complex processing algorithms. This leads to delays in each iteration and hardships in establishing an efficient communication between team members and the clients.

On the other hand, traditional methodologies have a well-defined requirements model before the implementation and coding process starts and this acts as a reference for the development team during the coding process. Clients do not participate in this stage of the development life cycle. The development team will perform the coding according to the documentation provided by the business analysts until the system is complete and only then it will be presented to the clients as final product. Developers are not concerned about frequent meetings and have more time to finish the system. This allows them to provide a better product.

The fact that agile development allows changes in requirements in an incremental way lead to two dependency problems in design: *rigidity* and *mobility*. Rigidity means a change in the system leads to a cascade of changes in other modules, while mobility means the inability of the system to include reusable components because they involve too much effort or risk. When such problems are present throughout the system, there must be a high level restructuring in order to eliminate unwanted dependencies. [10] Table 1 sum up the differences between agile and traditional approaches.

Table 1. Differences between traditional and agile development [11] [12] [13]

	Traditional development	**Agile development**
Fundamental hypothesis	Systems are fully specifiable, predictable and are developed through extended and detailed planning	High quality adaptive software is developed by small teams that use the principle of continuous improvement of design and testing based on fast feed-back and change
Management style	Command and control	Leadership and collaboration
Knowledge management	Explicit	Tacit

DOI: 10.12948/issn14531305/17.4.2013.06

	Traditional development	**Agile development**
Communication	Formal	Informal
Development model	Life cycle model (waterfall, spiral or modified models)	*Evolutionary-delivery* model
Organizational structure	Mechanic (bureaucratic, high formalization), targeting large organization	Organic (flexible and participative, encourages social cooperation), targeting small and medium organizations
Quality control	Difficult planning and strict control. Difficult and late testing	Permanent control or requirements, design and solutions. Permanent testing
User requirements	Detailed and defined before coding/implementation	Interactive input
Cost of restart	High	Low
Development direction	Fixed	Easily changeable
Testing	After coding is completed	Every iteration
Client involvement	Low	High
Additional abilities required from developers	Nothing in particular	Interpersonal abilities and basic knowledge of the business
Appropriate scale of the project	Large scale	Low and medium scale
Developers	Oriented on plan, with adequate abilities, access to external knowledge	Agile, with advanced knowledge, co-located and cooperative
Clients	With access to knowledge, cooperative, representative and empowered	Dedicated, knowledgeable, cooperative, representative and empowered
Requirements	Very stable, known in advance	Emergent, with rapid changes
Architecture	Design for current and predictable requirements	Design for current requirements
Remodeling	Expensive	Not expensive
Size	Large teams and projects	Small teams and projects
Primary objectives	High safety	Quick value

6 Agile Software Development Methods

There are several agile software development methods available. [14] Although each of them has a unique approach, they share the values and visions described by the *agile* manifesto. They all involve permanent communication, planning, testing and integration. They help develop good software, but what defines agile methods is that they encourage collaboration and makes common decisions good and fast.

Some of the agile software development methodologies are: Adaptive Software Development (ASD), Feature Driven Development (FDD), Crystal Clear, Dynamic Software Development Method (DSDM), Rapid Application Development (RAD), SCRUM, Extreme Programming (XP) and Rational Unify Process (RUP). Some of them are described in Table 2. In time several methods based on some of these values and principles have emerged. They have been designated as agile, and some of them are:

- Extreme programming (XP);
- Scrum;

DOI: 10.12948/issn14531305/17.4.2013.06

- Agile unified process (AUP);
- Agile modeling;
- Essential unified process (EssUP);
- Open unified process (OpenUP);
- Dynamic system development method (DSDM);
- Feature driven development (FDD);
- Crystal.

Table 2. Main agile development methods with key references [14]

Agile method	Description	Reference
Crystal methodologies	A family of methods for teams of various sizes: *Clear, Yellow, Orange, Red, Blue*. The most agile methods, *Crystal Clear*, concentrate on communication between small teams that develop non-critical software. Development has seven characteristics: frequent delivery, reflexive improvement, osmotic communication, personal safety, concentration, easy access to expert users and requirements for technical environment.	[15]
Dynamic software development method (DSDM)	Divides projects into 3 stages: pre-project, project life cycle and post-project. DSDM is based on nine principles: users involvement, empowering the project team, frequent delivery, approaching current needs of the business, iterative and incremental development, allows reversing the changes, high end goal is established before project starts, testing during the life cycle, efficient communication.	[16]
Feature-driven development	Combines model driven development with agile development, emphasizing the initial object model, work division into features and iterative design of each feature. Claims to be best suited for critical system development. An iteration of a feature has two stages: design and development.	[17]
Scrum	Concentrates on project management, for situations where initial planning is difficult, with mechanisms for "empiric process control", where feed-back loops are the main element. The software is developed by a team (that self organizes) in stages (called "sprints"), starting with planning and ending with assessment. The features that must be implemented are recorded in a list of unsolved orders. The client decides which orders are to be implemented in the next sprint. Team members coordinate their activity in daily briefings (at the beginning of the day). One member (chief/master scrum) is responsible with solving issues that prevent the team from working efficiently.	[18]
Extreme programming (XP; XP2)	Concentrates on the best development practices and consists of 12 stages/activities: planning game, small launches, metaphor, simple planning, testing, refactoring, peer programming, collective ownership, continuous integration, 40 hour week, on-site clients and coding standards. The revised version, XP2, consists of the following primary practices: whole team, informative work space, work under pressure, peer programming, stories, weekly cycle, trimester cycle, 10 minute development, continuous integration, incremental design etc. There are also 11 additional practices.	[19, 20]

The agile group of methods is based on iterative and incremental development, where specifications and solutions come from collaboration between teams individually organized, but with a common goal. These methods are based on 12 principle, synthesized in the so called **Agile Manifesto** published in 2001 [21]:

- client satisfaction, through rapid delivery of usable software;

DOI: 10.12948/issn14531305/17.4.2013.06

- meeting the specifications, even if it happens late in the development;
- frequent delivery of usable software (weekly);
- usable software is the main measure of progress;
- sustained development, able to keep a steady rhythm;
- cooperation between developers and clients;
- face-to-face cooperation is the best way to communicate;
- projects are built by motivated and credible persons;
- simplicity;
- individually organized teams;
- adaptation to changing circumstances;
- permanent attention to excellent technique and good design.

SCRUM is an iterative and incremental method whose purpose is to help development teams to concentrate on established goals and minimize the work done on less important tasks. SCRUM aims to keep the simplicity in a complicated business environment. The term comes from rugby, where it is a strategy to return a lost ball into the game by team work. SCRUM does not provide implementation level techniques; it focuses on the way the members a development team should interact to create a flexible, adaptive and productive system in a constantly changing environment. The method was presented in details by Schwaber and Beedle. SCRUM is based on two elements: *team autonomy* and *adaptability*. Team autonomy means that project leaders establish the tasks the team must perform, but in each iteration the team is free to decide how to work, with the goal of increasing team productivity. SCRUM does not propose specific software development techniques, but method sand instruments regarding the management for various phases, in order to avoid the confusion created by project complexity and unpredictability.

"XP is an easy methodology for small to medium teams that develop software products with vague or changing requirements" is the definition given by Kent Beck, the creator of extreme programming [22]. **Extreme programming** (XP) is a modern development model, inspired from RUP. Program development does not mean hierarchies but collaboration within the team. Team members are encouraged to assert their personality and offer and receive knowledge and become great programmers. XP considers that program development means first of all writing programs. It is suited for projects with dynamic requirements or those that are not well defined from the start. There must be a partnership between client and programmers. It does not generate too much documentation. XP describes four main activities: coding – main activity; testing – every module must be tested; listening – the programmer must communicate with the client in order to understand his needs; design – building a correct architecture of the system will lead to an efficient system and reduce unnecessary dependencies between modules.

7 Conclusions

No matter what model is chosen for developing software applications, this activity involves complex processes that are often predisposed to errors. That is why, beyond agility or traditionalism, an important role goes to testing and validation. Any high quality software system, with professional development and implementation must be tested and validated before going into production. The client must know that the system was developed and implemented according to the project specifications. Also, the client must be sure the project functionality is correct (to be continued).

References

[1]http://www.techopedia.com/definition/22193/software-development-life-cycle-sdlc
[2]http://www.tutorialspoint.com/sdlc/sdlc_overview.htm
[3] http://istqbexamcertification.com/what-are-the-software-development-models/
[4]http://www.slideshare.net/J.T.A.JONES/software-development-life-cycle-model-

DOI: 10.12948/issn14531305/17.4.2013.06

1392777
[5]http://www.tutorialspoint.com/sdlc/sdlc_overview.htm
[6] J. Bloomberg, Why service-oriented management? 2002, SearchSOA.com. Retrieved September 15, 2011, from http://searchsoa.techtarget.com/tip/Why-service-oriented-management
[7] D. Vesset, M. Fleming, H. Morris, S. Hendrick, B. McDonough, S. Feldman, E. Traudt, C. Olofson and M. Webster, Worldwide Decision Management Software 2010–2014 Forecast: A Fast-Growing Opportunity to Drive the Intelligent Economy. IDC. Retrieved September 15, 2011, from http://www.idc.com/research/viewdocsynopsis.jsp?container Id=226244
[8] L.R. Vijayasarathy, Agile Software Development: A survey of early adopters. Journal of Information Technology Management Volume XIX, Number 2. 2008
[9] K. Peterson, A Comparison of Issues and Advantages in Agile and Incremental Development between State of the Art and an Industrial Case. Journal of System and Software. 2009
[10] G.K. Henssen, Maintenance and Agile Development: Challenges, Opportunities and Future Directions. 2009
[11] S. Nerur, R. Mahapatra, G. Mangalaraj, Challenges of migrating to agile methodologies, Communications of the ACM (May) (2005) 72– 78
[12]http://www.producao.ufrgs.br/arquivos/disciplinas/507_artigo_3_empirical_systematic_revi ew.pdf
[13] Get Ready for Agile Methods, with Care,Barry Bohem, IEEE Ianuarie 2002, tabel 1, pg. 68
[14]http://www.slideshare.net/J.T.A.JONES/software-development-life-cycle-model-1392777
[15] A. Cockburn, Crystal Clear: A Human-Powered Methodology for Small Teams, Addison-Wesley, 2004, ISBN 0-201-69947-8
[16] J. Stapleton, DSDM: Business Focused Development, second ed., Pearson Education, 2003, ISBN 978-0321112248
[17] S.R. Palmer, J.M. Felsing, A Practical Guide to Feature-driven Development, Prentice Hall, Upper Saddle River, NJ, 2002, ISBN 0-13-067615-2
[18] K. Schwaber, M. Beedle, Agile Software Development with Scrum, Prentice Hall, Upper Saddle River, 2001
[19] K. Beck, Extreme Programming Explained: Embrace Change, Addison-Wesley, 2000, ISBN 0-201-61641-6
[20] K. Beck, Extreme Programming Explained: Embrace Chage, second, ed., Addison-Wesley, 2004, ISBN 978-0321278654
[21]http://share.pdfonline.com/af77ce72a5594026b7be919e08f6e6b8/agile.htm
[22] Kent Beck 2000. Extreme Programming Explained Addison-Wesley Publishing Co

Marian STOICA received his degree on Informatics in Economy from the Bucharest University of Economic Studies in 1997 and his doctoral degree in economics in 2002. Since 1998 he is teaching in Academy of Economic Studies from Bucharest, at Economic Informatics and Cybernetics Department. His research activity, started in 1996 and includes many themes, focused on management information systems, computer programming and information society. The main domains of research activity are Information Society, E-Activities, E-Working, and Computer Science. The finality of research activity still today is represented by over 50 articles published, 9 books and over 20 scientific papers presented at national and international conferences. Since 1998, he is member of the research teams in over 15 research contracts with Romanian National Education Ministry and project manager in 5 national research projects.

DOI: 10.12948/issn14531305/17.4.2013.06

Marinela MIRCEA, associate professor, PhD, currently working with Bucharest University of Economic Studies, Faculty of Cybernetics, Statistics and Informatics, Department of Economic Informatics and Cybernetics. Competence areas: information system, Business Intelligence. Research in the fields of in-formation system, Business Intelligence, classification techniques. Author of 6 books and more than 50 papers published in national and international journals.

Bogdan GHILIC-MICU received his degree on Informatics in Economy from the Bucharest University of Economic Studies in 1984 and his doctoral degree in economics in 1996. Between 1984 and 1990 he worked in Computer Technology Institute from Bucharest as a researcher. Since 1990 he teaches at Bucharest University of Economic Studies from Bucharest, at Economic Informatics and Cybernetics Department. His research activity, started in 1984 includes many themes, like computers programming, software integration and hardware testing. The main domain of his last research activity is the new economy – digital economy in information and knowledge society. Since 1998 he managed over 25 research projects like System methodology of distance learning and permanent education, The change and modernize of the economy and society in Romania, E-Romania – an information society for all, Social and environmental impact of new forms of work and activities in information society.

DOI: 10.12948/issn14531305/17.4.2013.06