Articles

Software Development Methodologies for Reducing Project Risks

Veselin Georgiev[*]

Kamelia Stefanova[**]

Summary:

The rapid expansion of information and communication technologies in all business and social areas today has increased the pressure on and the need for the optimization of software development process. More than ever, the relation between quality, cost and time for delivery become a critical requirement for success. Developing cost competitive software products that meet high quality standards according to the time constrained market today is becoming a demanding task. Most of the software development processes are overburdened with strict and heavy documentation procedures that require strong control mechanisms and create additional sophistication in the software projects management. New families of methodologies for software development, referred to as Agile Processes, are introduced to meet the challenges of the future developments in the software industry. These methodologies focus on the flexibility and adaptability and are described as agile, unlike the traditional processes which are restrictive in the introduction of changes. The application of these methodologies and practices answer the need to reduce risks and increase quality and usability of the final software product.

Key words:

Agile Software development, Project risks, SCRUM, Extreme programming, Feature Driven Development, Adaptive Software Development

JEL Classification: *C61, C63, C8, D81*

Introduction

The vigorous growth of the information technology in recent years has created prerequisites for the introduction of new models in the management of software projects. A characteristic feature of the process of software development is that it is usually dependent on other software solutions and products in a production plan. This dependence in an environment of dynamic development requires continuous renewal and compliance compared to other technologies. For large projects it is even more important. Also distinctive entrepreneurial culture in this area is gaining more popularity globally, which is reflected in the increasing number of companies providing software solutions. The investment environment is very dynamic, and the development of new software solutions is a target for most investors and investment funds. This directly affects the possibility of developing competitive software solutions and products.

[*] PhD Student, Department Information Technologies and Communications, University of National and World Economy, email: vgeorgiev@petrovkata.com, twitter: @petrovkata

[**] Professor, PhD, Department Information Technologies and Communications, University of National and World Economy

Agile Development Methodologies are becoming an increasingly popular means of minimizing the risks associated with the planning process and development of software solutions. Their iterative nature of performance, openness to constant change, the close cooperation between the customer and the developers are the key features that meet the modern business needs.

1. Software development risks

The process of software development is characterized by specific risks that accompany various stages of the life cycle. Depending on the chosen methodology and workflow of the software development solution, these risks can have varying rate of occurrence or have different character. The term software risk refers to unexpected events that may cause a negative impact on the software product, the duration needed for its development and the resources used. Risks can be addressed in two main areas: risks external to the organization over the occurrence of which cannot be influenced; internal risks, which are dependent on the organizational workflows.

- **External Risks** - External events are mainly outside of the project manager control and, in most cases, the corporation. Examples include:
  - Marketplace developments — rapid developments can cause an abrupt change of direction
  - Government regulatory changes
  - Industry-specific procedures—new standards, issues
  - Mergers/acquisitions
  - Legal issues—disputes, lawsuits, and court orders
  - Change-driven factors—new products, services, changes in market
  - Corporate strategy and priority changes
  - Disasters such as fire, flood, earthquake, or other natural disaster
  - Disruptions caused by interference from external electrical sources
  - Loss of power leading to heating or ventilation malfunctions; air conditioning failures
  - Sabotage, hacking and security breaches
  - Communications systems and security sensor failures
  - Viruses and other malicious attacks on information systems
  - Emergency destruction of communications
  - Platform versions updates during development cycle

Most of these risks are very difficult to control at the project manager level but can be identified and, accordingly managed. This means that senior management should be involved in the risk management process and take considerations.

- **Cost Risks** - Most of these types of risks are directly or indirectly under the project manager's control or within the area of influence. Examples of cost risks include those arising from:
  - Cost overruns by project teams or subcontractors, vendors, and consultants
  - Scope creep, expansion, and change that has not been managed
  - Poor estimation or errors that result in unforeseen costs
  - Overrun of budget and schedule.
- **Schedule Risks** - Schedule risks can cause project failure by missing or delaying a market opportunity for a product or service. Such risks are caused by:
  - Inaccurate estimation, resulting in errors
  - Increased effort to solve technical, operational, and external problems
  - Resource shortfalls, including staffing delays, insufficient resources, and unrealistic expectations of assigned resources

- Unplanned resource assignment—loss of staff to other, higher-priority projects.

- **Technology Risks** - Technology risks can result from a wide variety of circumstances. The result is a failure to meet systems' target functionality or performance expectations. Typical examples are:
  - Problems with immature technology
  - Use of the wrong tools
  - Software that is untested or fails to work properly
  - Requirement changes with no change management
  - Failure to understand or account for product complexity
  - Integration problems
  - Software/hardware performance issues - poor response times, bugs, errors.
- **Operational Risks** - Operational risks are characterized by an inability to implement large-scale change effectively. Such risks can result in failure to realize the intended or expected benefits of the project. Typical causes are:
  - Inadequate resolution of priorities or conflicts
  - Failure to designate authority to key people
  - Insufficient communication or lack of communication plan
  - Size of transaction volumes - too great or too small
  - Rollout and implementation risks - too much, too soon.

One of the key elements in the process of management of the wide range of risks and their mitigation is the choice of the right methodology for managing the entire software development process. There are dozens of methodologies that have been applied to the development of modern software products in recent years, but the most productive, suitable and widely used are the Agile methodologies for software development. This topic has been already studied (e.g. John McManus, 2004)

2. Agile Software Development practices

The Manifesto for Agile Software Development, published in 2001 (by Agile Alliance), defines the agile methodologies for software development.

Over the next few years, these methodologies emerge as a natural reaction to traditional software development approaches. The reason for this is the recognized need for an alternative to the heavy documentation processes. There are many agile methodologies for software development. Their common features are related to the need constantly to adapt the changing conditions and requirements that accompany a software project. They comprise a well-known set of practices, but combined in such a way so as to bring the project to a successful completion. Not all of them cover the entire development process, some focus only on certain stages. Rarely is the same methodology applicable universally in every area in which we need a software solution. Sometimes even the same methodology is not suitable twice consecutively. The successful implementation of a combination of different methodologies with their practices would lead to the successful establishment of its own process within a software project.

- **Extreme Programming** (XP - Fig. 1) develops as a consequence of the problems that have arisen in the development cycle of standard models and processes. Started simply as "an opportunity to get the job done" practices that have been proven effective in the development of software in previous years. After a series of successful projects in XP practice is "theorized" in its essential principles by Kent Beck in 1999. Although individual XP practices

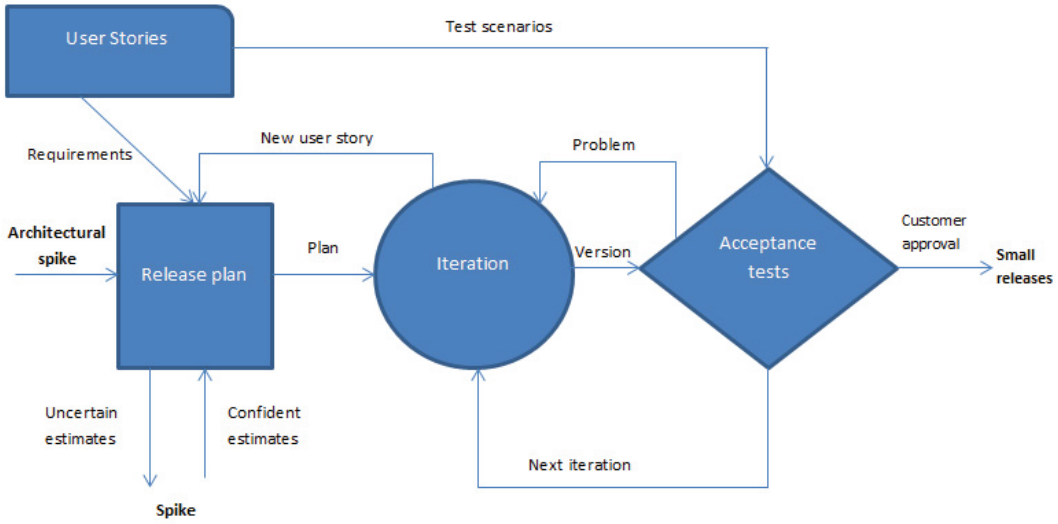

Fig. 1. Methodology process in Extreme Programming

are not new, they are collected and combined to function together in a new way, forming a new methodology for software development. It is a system of practices that the community of developers evolutionarily evolved to cope with problems such as rapid delivery of quality products, meeting the needs of business. The term "extreme" comes from the fact that these conventional practices are applied to the extreme level. On Fig. 1 can be seen the main phases and practices through which an XP project passes.

Extreme Programming is created for the use mainly by small, co-located teams developing software with requirements that were not clearly defined or are changing. This methodology emphasizes on the team work and customer involvement throughout the software development. Communication and feedback are focal points in Extreme Programming. Communication is advocated not just between the customer and the analyst, but also between the customer and the developers, and within the development team itself. The feedback is achieved through early and continuous testing. Proponents of Extreme Programming assert that its benefits include faster time to market, higher quality software development, better customer satisfaction, and highly motivated working teams.

The basic practices of Extreme Programming are pair programming, refactoring, testing, continuous integration, and evolutionary design. It should be noted that there are many other practices advocated in Extreme Programming and that the core practices of Extreme Programming have been named and renamed by many authors.

- **Pair Programming -** is the practice of two programmers/engineers participating in the development effort of one programming unit. Typically, the pair is divided into one person entering code or test cases while the other is reviewing and thinking. The code resulting from pair programming is more defect free, does not take significantly more time to develop than if developed by one member, yields fewer lines of code, and is more sat-

isfying to programmers. Pair programming encourages the design of the programming unit to evolve simultaneously with the actual programming.

- **Refactoring** – is an activity of continuous re-design of a program unit to take advantage of programming techniques, especially object-oriented design and design patterns, to make the programs more reusable, simpler, and more efficient. Refactoring can occur at various times throughout the development process. A goal of refactoring is to yield programming units with a strong internal structure. Additionally, refactoring allows developers to respond quickly to a change in customer requirements or technology.
- **Testing** - by Extreme Programming, includes ideas such as unit testing by programmers, functional testing by users, and automated testing to generate test units that mirror the actual programming units. These concepts fully support the testing of programming units to promote a defect free unit with each unit release. Testing in Extreme Programming involves two types of tests. The first type of testing is unit testing which is the testing of individual programming units such as a class. Developers write tests for every class they produce and even write the tests before writing the code. When they add functionality to the original class, they can test the prior functionality with the developed testing unit and add more tests for the new functionality. The second type of testing in Extreme Programming is functional testing. Functional tests are scripts developed to test clusters of classes. Functional testing is typically use case driven tests. These tests are responsibility of the users or customers while the unit tests belong to the developers.
- **Continuous Integration** - is a concept of integrating new code into existing code and then utilizing the testing techniques defined by Extreme Programming. This practice yields units of code that are continually tested during the development.
- **Evolutionary Design** - involves making iterations of a program within

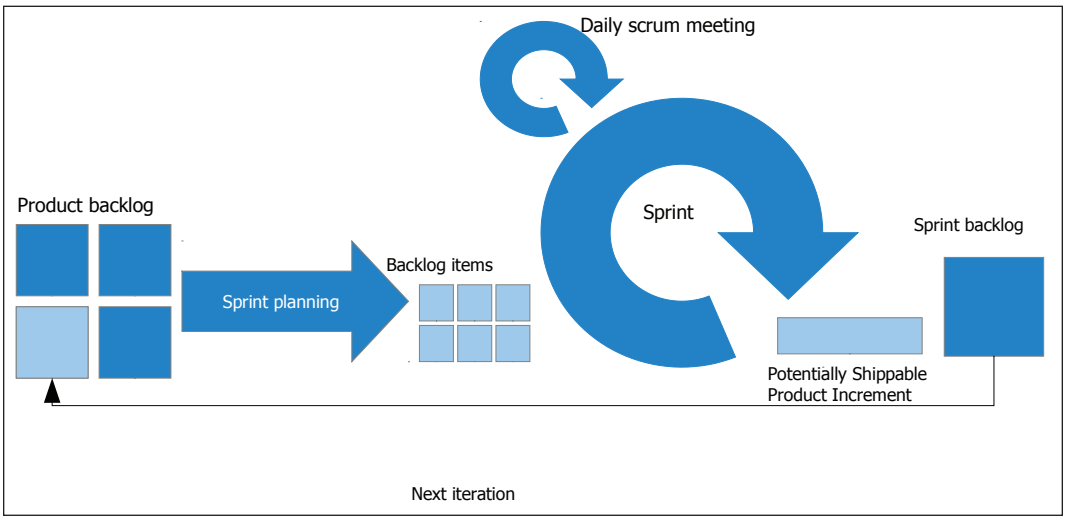

Fig. 2. Process in software development with Scrum

minutes rather than days. The programming pair defines each iteration of the problem and then implements these iterations. Upon completion, the problem is expanded to yield another iteration and then that iteration is implemented.

Extreme programming has been already studied by Smith and Stoecklin (2001) and Hneif and Hock Ow (2010)

- **Scrum** is another popular agile methodology, the term comes from the game of rugby and means clustering of players around the ball in order to move it forward. This agile approach (Fig. 2) is established to manage the process of software development. It is the "empirical approach employing concepts of the theory of control of industrial processes during the development of the software, as a result of which exhibit the ideas of flexibility, adaptability and productivity." The process enables software companies to realize their projects quickly and immediately begin to put into operation. Scrum focuses on the way the team members work together to build a flexible system in a constantly changing environment. Scrum helps to improve existing practices in the organization, emphasizing on the frequent intervention of management to identify problems and obstacles during the process. This topic has been already studied (H. J. Correa, 2008).
- **Feature Driven Development - FDD** is a flexible and adaptive approach to developing software systems. It does not cover the entire software process, but rather focuses on the phases of design and implementation. FDD combines iterative method of development best practices proven to be effective in the industry. Emphasizes on quality in the process and often provides the functionality of the client as well as continuous monitoring of the progress. The task for each function is to be described and defined so as to enable the customer to consider its value and decide whether or not to include it in the product.

FDD consists of five sequential processes and provides methods, techniques and guidelines required by participants in the project to build the system. Also, FDD includes roles, goals and schedules needed for the project. It is claimed that, unlike some other methodologies, FDD is suitable to the development of business critical systems, considered by Coad (2002) and Goyal (2007)

- **Adaptive Software Development (ASD)** (Fig. 3) is developed by James A. Highsmith (2000). ASD focuses primarily on the development of large and complex systems. This methodology encourages the development of the project through constant development of prototypes. Overall, ASD is trying to "balance on the edge of chaos" - its purpose is to set a framework with sufficient guidance for the project not to fall into chaos, without restricting the self-expression and creativity of the developer.

There are three main components in Adaptive Software Development:

- The Adaptive Conceptual Model
- The Adaptive Development Model
- The Adaptive Management Model

Without practical techniques, conceptual ideas remain only as concepts. On the other hand, techniques without a theoretical base are reduces to a series of steps executed by rote. Concepts and practices reinforce each other as extreme projects do not succumb to rote practices, they require judgment based on firm conceptual foundations.

Extreme environments move quickly, demanding fast learning among project members and often forcing people to abandon preconceived assumption. Fast learning requires iterations – try, review, repeat. Accelerated schedules demand a

high degree of concurrency with developers working on many components at the same time. Taken together, iteration and concurrency generate high levels of change, especially as project size escalates.

The Adaptive Conceptual Model introduces the new science of complex adaptive systems as the conceptual foundation for both development and management. The Adaptive Development Model focuses on iterative phases of development and work-group-level practices to increase speed and flexibility. The Adaptive Management Model focuses on forging adaptive culture and identifying adaptive practices, particularly those involving distributed work groups and dealing with high level of change, collaboration and management results.

Agile software methodologies has been studied too by Paetsch, Eberlein, Maurer (2003), Larman (2004), Todorov (2014).

The Agile practices differ from the traditional ones mainly in the development process stages. Authors single out 14 which should be differentiated, and focus on the management model, communication patterns approaching final goals:

Agile methodologies characteristics compared to the traditional approaches

- **Development lifecycle** – traditional approach uses linear development life-cycle model rather than agile where the process includes iterative and evolutional delivery model
- **Style of development** – for traditional models is typical to be used expectant type and adaptive approach for agile practices
- **Requirements** – traditional models expect good documented and knowable development process rather than agile methods where process can include rapid or emergent change, or unknown issues to be discovered during the project.

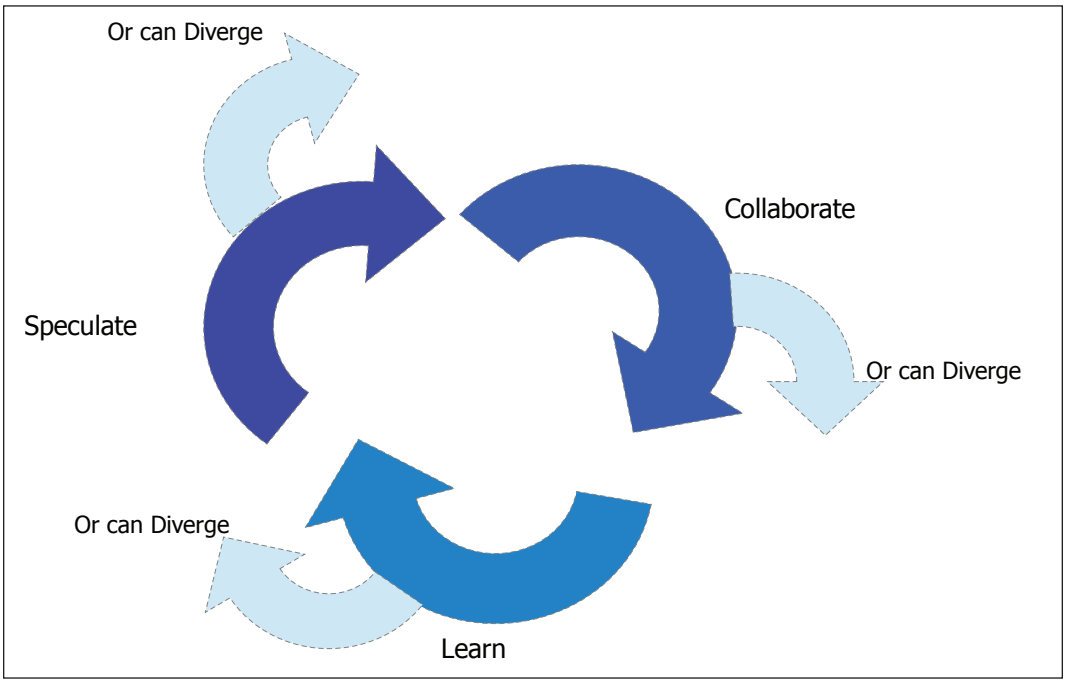

Fig. 3. Adaptive Software Development process [7]

- **Architecture** - Heavyweight architecture for current and future requirements is typical for traditional approach, while agile practices in fact do not need it, this is known as acronym (YAGNI), which is typical in agile development methods where whole process in documentation or architectural model is not needed at all.
- **Management** – the whole process in traditional models are process-centric and the process is operated by command and control. For agile practices the process is people-centric operated by leadership and team collaboration
- **Documentation** – heavy and detailed documentation for traditional models, which gives explicit knowledge. In agile approach there is typical light documentation and face-to-face communication
- **Goal** – predictability and optimization for traditional models versus exploration and adaptation for agile practices
- **Change** – hard to change process and project for traditional models and easy and open for change project and process for agile models
- **Team members** - Pre-structured teams for traditional approach versus Self-organizing teams in agile models
- **Client Involvement** – low involvement of clients in the development process for traditional models. Client is involved in the beginning of the project, during the planning. In agile models client is involved as part of the team.
- **Organization culture** – command and control culture for traditional approach versus leadership and collaboration for agile practices
- **Software development process** - Universal approach and solution to provide predictability and high assurance is typical for traditional approach. Flexible approach adapted with collective is typical for agile models.

This topic has been already studied by Moniruzzaman and Hossain (2013)

Based on the main differences between traditional and agile practices some observations about risk changes for major agile practices can be pointed out. Depending on the project application of agile methodologies, the practices may vary as not always the same practice is applicable to similar projects, however we can outline the main advantages of practices in main risk areas in order to mitigate and manage risk:

Agile process application to main risk categories

Conclusion

Flexible software methodologies and practices for software development provide developers with freedom in the process of work and interaction with customers and owners of the products. This approach contributes to the improvement of the quality of the final product and minimizes the risks associated with customer expectations. Agile methodologies are not best suited for all projects. When communication between the developer and the customer is difficult, or when the development team includes mainly beginners, agile methodologies will not yield the best possible results. The process of continuous planning and incremental approach is a successful tool to minimize operating risks and risks associated with planning, technological risks and those related to changes in the environment and the actions of competitors and customers.

Table 1. Main Software development risks / Agile development practices application

Risk Categories	Note	Result in risk change
External Risks	Short periods of planning and iterative nature make it possible to take account of changes in the external environment that will enable taking the appropriate action and limiting their consequences for the software product	Risks are reduced
Cost Risks	The ability to target a small number of elements of the entire project in every iteration, facilitates the more accurate assessment of the resources required to build the product. This increases the cost for stage management and planning of the project, but at a much lower risk of major discrepancies in the actual and predicted cost of the project	Risks are reduced, Total cost is increased
Schedule Risks	Smaller portions planning allow more precise definition of the terms and tasks to be performed. Larger count of iterations and the inclusion of the product owner in the discussions can lead to adding unforeseen additional iterations or design changes, which increase the time needed for full software development. This in turn increases the quality of the final product and satisfies the expectations of product owner.	Schedule risks are increased
Technology Risks	Agile methodologies enable developers to explore, experiment and test different technologies to achieve the highest quality of software products. This in turn reduces the risk in choosing wrong technologies. The iterative nature of development and testing reduces the risk to omit testing any part of the project.	Technology risks are reduced
Operational Risks	Operational risks are typical when developing large applications or implementing major changes to existing applications, due to the risk of losing or not well planned part of the project. Typical for agile methodologies is the slicing of the whole process into small portions of the development, which provides for the closer examination of every aspect of the project, and for minimizing the risk of non-good performance or quality	Operational risks are reduced

Flexible approaches also allow for regular team meetings, which is a good tool for troubleshooting related to teamwork, and personal and individual problems of each participant. This type of practice needs to target a larger budget, due to the longer period of development. The increased number of iterations, meetings and additional planning require more resources planned for the product's implementation, though

there is a smaller risk of budget change compared to the projects implemented with traditional methodologies. Flexible practices for development as a whole can be characterized as a good practice for managing the dynamics of business processes, distributed teams changing business requirements or environment by enabling the products to meet demands.

References

Agile Alliance, 2009. Manifesto for Agile Software Development. Available at: http://www.agilemanifesto.org [Accessed February, 8th, 2014]

Coad P., Feature-Driven Development. Available at: http://www.petercoad.com/ download/bookpdfs/jmcuch06.pdf [Accessed February, 12th, 2014]

Correa H. J., 2008. Introduction to Scrum

Everette R. Keith, 2002 . Agile Software Development Processes. A Different Approach to Software Design

Goyal S., 2007. Feature Driven Development Agile Techniques for Project Management and Software Engineering, Technical University Munich. Available at: http://csis.pace.edu/~marchese/CS616/Agile/FDD/fdd.pdf [Accessed March, 5th, 2014]

Highsmith J. A., 2000. Adaptive Software Development: A Collaborative Approach to Managing Complex Systems. Available at: http://books.google.ca/books?id=CVcUAAAAQBAJ&lpg=PP1&hl=bg&pg=PR15#v=onepage&q&f=false [Accessed December, 12th, 2013]

Highsmith J. A., 2011. Don't plan, speculate

Hneif M., Siew Hock Ow, October 2010 , Review of agile methodologies in software development. Available at: http://www.arpapress.com/Volumes/Vol1/IJRRAS_1_01.pdf [Accessed March, 9th, 2014].

Larman C., 2004. Agile and Iterative Development: A Manager's Guide. Available at: http://books.google.bg/books?id=e4FrAFn0ytIC&printsec=frontcover&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false [Accessed March, 9th, 2014]

McManus J., 2004. Risk Management in Software Development Projects. Available at: http://www.google.bg/books?id=l_VPK9wUJGIC&printsec=frontcover&hl=bg#v=onepage&q&f=false [Accessed March, 11th, 2014]

Moniruzzaman A B M, Dr Syed Akhter Hossain, 2013 . Comparative Study on Agile software development methodologies

Murch R., 2000. Project Management: Best Practices for IT Professionals. ISBN-10: 0-13-021914-2

Paetsch F., Eberlein A., Maurer F., 2003. Requirements engineering and agile software development, Enabling Technologies: Infrastructure for Collaborative Enterprises. WET ICE 2003, ISBN 0-7695-1963-6, 2003, p. 308-313

Smith S., Sara Stoecklin, 2001. What We Can Learn From Extreme Programming. Available at: http://faculty.salisbury.edu/~xswang/research/papers/serelated/xp/p144-smith.pdf [Accessed March, 20th, 2014]

Todorov N., 2013. Agile Methodologies for software development and application base on standards in quality management. Bulgarian Academy of Science