Comparative study on software development methodologies

Mihai Liviu DESPA
mihai.despa@yahoo.com

Bucharest University of Economic Studies

This paper focuses on the current state of knowledge in the field of software development methodologies. It aims to set the stage for the formalization of a software development methodology dedicated to innovation orientated IT projects. The paper starts by depicting specific characteristics in software development project management. Managing software development projects involves techniques and skills that are proprietary to the IT industry. Also the software development project manager handles challenges and risks that are predominantly encountered in business and research areas that involve state of the art technology. Conventional software development stages are defined and briefly described. Development stages are the building blocks of any software development methodology so it is important to properly research this aspect. Current software development methodologies are presented. Development stages are defined for every showcased methodology. For each methodology a graphic representation is illustrated in order to better individualize its structure. Software development methodologies are compared by highlighting strengths and weaknesses from the stakeholder's point of view. Conclusions are formulated and a research direction aimed at formalizing a software development methodology dedicated to innovation orientated IT projects is enunciated.

Keywords: *software development, project management, development methodology*

1 Introduction

The management process of a software development projects follows the basic rules of project management but also includes particular features. A software development project manager has to deal with challenges and setbacks that are proprietary to the IT industry. Also in software development there are benefits and strong points that help ease the burden of management.

Software development projects are notorious for frequently changing initial planning and specifications. In order to identify the main reasons for changing specification during the development stage of a software product debates were started on LinkedIn project management groups. Debates were initiated on 11 LinkedIn groups starting the discussion with the following introduction: *Software development projects are notorious for frequently changing initial planning and specifications. What do you believe are the main reasons for changing specifications during the development stage?* Project managers responded on 3 out of the 11 groups. The 3 groups are: PMO – project management office, Agile Project Management Group and International Society for Professional Innovation Management. Based on the information collected from the LinkedIn discussions and on the author's own experience as software development project manager specifications change due to the fact that:

- the project owner identifies new business opportunities and decides to integrate them into the software being developed;
- due to the technical nature of software development projects there is a lack of shared understanding of

expected outcomes;
- original planning was based on specifications that were misinterpreted by the project manager or poorly illustrated by the project owner;
- project team is unable to implement planned functionalities due to lack of expertise or technological limitations;
- the context in which the software is going to be used changes thus generating the need for the software to change;
- new technology or software product is launched on the market.

Changing specifications has a negative impact on the project management process as it reduces predictability and exercises pressure on the budget and deadlines.

The software development field is characterized by high dynamics of technology and standards. Programming languages evolve, new frameworks arise and fall with astonishing speed, user interfaces become more and more diverse as software is required to work on a larger array of devices. PHP server-side scripting language registered 16 releases on new and improved versions in 2014 alone [1]. The software development community widely accepted Phalcon and Laravel, as two of the most powerful PHP frameworks. Both were released in 2012. The project manager has to keep up with the latest trends in software development in order to meet the project owner's requirements and in order coordinate effectively the project team.

Software development projects involve project teams that are made up of highly skilled and highly trained individuals with predominantly technical backgrounds. Highly skilled and highly trained individuals will require significant compensation for their work and that will translate into high man-hour or man-day rates. So the project manager is under considerable pressure to provide accurate time estimates in terms of required man-days as every inconsistency will generate significant additional costs. Also highly skilled individuals don't integrate well in a team as they have a tendency of being arrogant and self-absorbed. The project manager should be able to exploit their ego in the best interest of the project and mitigate disputes and opinion clashes that occur between team members.

Software development projects are often implemented by teams that have members distributed all over the globe. Building software does not require formal face-to-face communication. Task assignment and task tracking is done by using online management tools like Pivotal Tracker, Basecamp or Producteev. Code version control is ensured by using versioning tools like SVN or GIT. File sharing is accomplished by using tools like Dropbox, Google Drive or Box. Online meetings can take place using Skype video conferences.

Table 1. Software development projects characteristics

Characteristic	Positive Impact	Negative Impact
frequently changing specifications	-	jeopardize deadlines
		results in exceeding the project budget
		causes stress and discontent for the development team
high dynamics of technology and standards	generates new opportunities in terms of design and codding	software can become obsolete by the time it hits the market
		software developers have to invest a lot of time in

		researching new technologies
skilled workforce	increases the likelihood of achieving innovative results	high cost generated by human resources
globally distributed teams	work can be performed around the clock	monitoring and control becomes more difficult
	cultural diversity nurtures creativity	integrating new code is more challenging

Table 1 summarizes the impact that software development characteristics have on the project management and implicitly on the project team. The only characteristic that does not have a positive impact is *frequently changing of specifications*.

2. Software development stages

Building a software product is a process consisting of several distinct stages. Each stage has its own deliverables and is bound by a specific time frame. Depending on the project, certain stages gain additional weight in the overall effort to implement the software product.

Research is the stage where the project owner, the project manager and the project team gather and exchange information. The project owner is responsible for formulating requirements and passing them on to the project manager. In order to properly formulate requirements the project owner has to first define a set of goals. Then he has to envision the way a software product will help him achieve those goals. In the research stage the project owner will try to find people or companies with similar goals and document the way those people or companies acted upon fulfilling their goals. The project manager is responsible for receiving the requirements from the project owner, evaluating them and passing them to the project team as technical specification. The project manager has to be able to evaluate the requirements from both a business perspective and a technical perspective. The project manager has to research market characteristics and user behaviour patterns. The project team is responsible for evaluating the requirements from a technical perspective. The project team will have to research the frameworks, API's, libraries, versioning tools and hosting infrastructure that will be required in order to build the software product.

Planning is the stage where all the elements are set in order to develop the software product. Planning starts with defining the overall flow of the application. Next step is to breakdown the flow into smaller, easier to manage subassemblies. For each subassembly a comprehensive set of functionalities has to be defined. Based on the required functionality a database structure is designed. Taking into account the overall flow of the application, the subassemblies, functionalities and database structure, the project manager together with the project team have to choose the technology that will be employed to develop the application. Also the project manager should decide on the best suited management methodology and the proper work protocol for the project at hand.

Design is the stage where the layout of the application is created. Web applications and mobile applications tend to grant more impotence to layout than desktop applications. Depending on the nature of the application designs can range from rough and functionality driven to complex and artistic. An accounting application will only require basic graphic design but an online museum will require high end design work. In an accounting application design has to emphasize and enhance functionality whereas in an online museum functionality has to be tailored in order to fit the design. The graphic design can overlap with the planning and with the programming stage. The graphic design stage is important because it will display to the project owner a preview of the application before it is actually built. At this stage usually the project owner comes up with new requirements that have to be summited to research and planning.

Development is the stage where code is written and the software application is actually built. The development stage starts with setting up the development environment and the testing environment. The development environment and the test environment should be synchronized using always the same protocol. Code is written on the development environment and uploaded on the test environment using the synchronization protocol. Another important aspect of the development stage is progress monitoring. The project manager has to determine actual progress and evaluate it against the initial planning. The project manager should constantly update the project owner on the overall progress. When writing code the software developers should also perform debugging operation in order to upload clean and bug free updates on the testing environment. Software developers should also comment their code so that they can easily decipher later or make it easy to understand for other developers.

Testing is the stage where programming and design errors are identified and fixed. Programming errors are scenarios were the application crashes or behave in a way it was not supposed to according to the designed architecture. Programing errors also consist in security or usability issues. If the application is vulnerable to attacks and can therefore allow attackers access to private data, then that is regarded as a programming error. If users have problem with slow response time from the application than that also is a programming issue. Design errors are actually inconsistency between what the project owner requested and what the project team ended up implementing. Design errors occur in the planning stage, have a significant impact on the project and are usually harder to fix. Identifying design errors is considerably more efficient when the project owner is involved as he is the one that formulated the application requirements.

Setup is the stage where the application is installed on the live environment. The setup stage precedes the actual exploitation of the software product. The setup entails configuring the live environment in terms of security, hardware and software resources. Back-up procedures are defined and tested. The actual setup of the software product includes copying the source code, importing the database, installing third party applications if required, installing cron-jobs if required and configuring API's if required. Once the application is installed it will go through another full testing cycle. When testing is completed content is added to the application.

Maintenance is the stage that covers software development subsequent to the application setup and also the stage responsible for ensuring that the application is running within the planned parameters. Ensuring that the application is running properly is done by monitoring the firewall, mail, HTTP, FTP, MySQL and SSH error logs. Also monitoring traffic data will provide valuable input on potential issues that may affect the application's performance. An important part of the maintenance stage consists of systematically testing functionalities for errors that were not identified in the testing stage or for issues that are not displayed in the error logs. The maintenance stage also provides the opportunity to add new features of functionality to the software application. Adding new code or changing the old code will have to be submitted to research, planning, programing, testing and setup.

The above mentioned stages are generally agreed by the software development community as being the cornerstones of every software development project. Depending of software development methodology they may be found under different naming conventions, they may be overlapping changing order or missing altogether.

3. Current software development methodologies

A software development methodology is a set of rules and guidelines that are used in the process of researching, planning, designing, developing, testing, setup and maintaining a software product. The methodology also includes core values that are upheld by the project team and tools used in the planning, development and implementation process. This paper reviews 20 of the most popular software development methodologies and highlights their core characteristics. The analysis includes specifying the scale of the project the methodology is suited for, the stage project owner feedback is delivered and a graphic representation of the methodology. For coherence reasons stages defined in the second section of the article are also used in depicting the graphic representations of the methodologies.

Waterfall is the first methodology generally acknowledged as being dedicated to software development. Its principals are for the first time described by Winston W. Royce even though the actual term waterfall is not used in the article [24]. It emphasizes meticulous planning and it outputs comprehensive documentation. The Waterfall methodology is linear-sequential process where every stage starts only after the previous has been completed. Each stage has its own deliverables. The Waterfall methodology is predictable and values rigorous software planning and architecture.

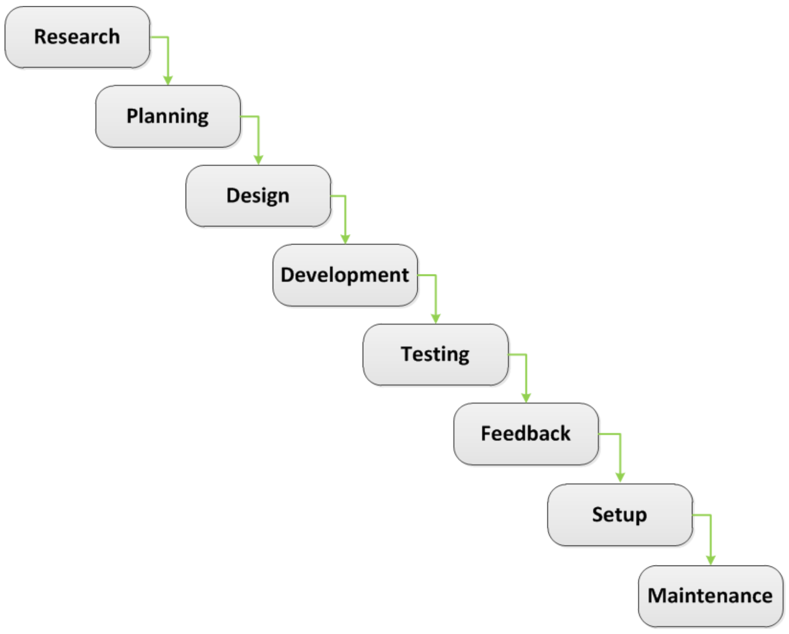


Fig. 1. Waterfall methodology

The project owner's feedback is received after the software application is completely developed and tested. The Waterfall methodology is suitable for small scale software development projects where requirements are clear and detailed planning can be easily drafted for the entire project.

Prototyping is a methodology that evolved out of the need to better define specifications and it entails building a demo version of the software product that includes the critical functionality**.** Initial specifications are defined only to provide sufficient information to build a prototype. The prototype is used to refine specifications as it acts as baseline for communication between project team and project owner. The prototype is not meant to be further developed into the actual software product. Prototypes should be built fast and most of the times they disregard programming best practices [2].

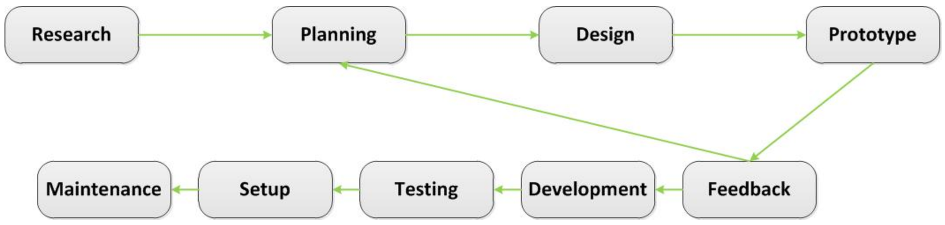


Fig. 2. Prototyping methodology

The project owner's feedback is received after the prototype is completed. The Prototyping methodology is suitable for large scale projects where is almost impossible to properly define exhaustive requirements before any actual codding is performed. Prototyping methodology is also suitable for unique or innovative projects where no previous examples exist.

Iterative and incremental is a methodology that relies on building the software application one step at the time in the form of an expanding model [3]. Based on initial specification a basic model of the application is built. Unlike the prototype, the model is not going to be discarded, but is instead meant to be extended. After the model is tested and feedback is received from the project owner specifications are adjusted and the model is extended. The process is repeated until the model becomes a fully functional application that meets all the project owner's requirements.

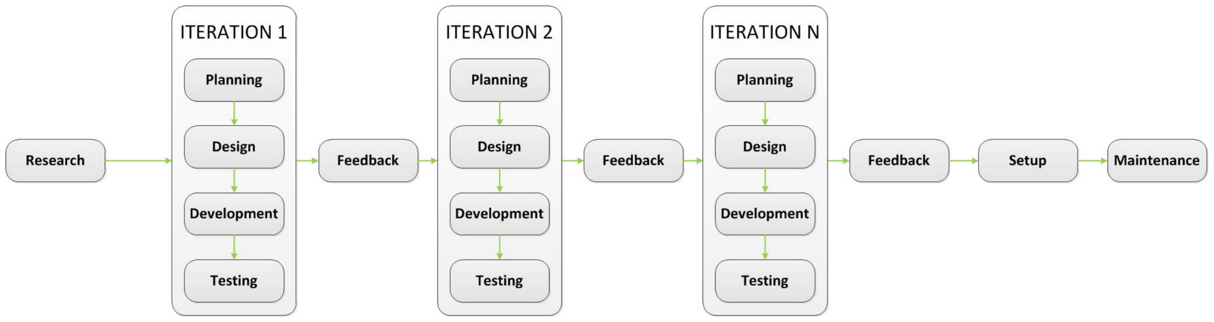


Fig. 3. Iterative and incremental methodology

The project owner's feedback is received after each iteration is completed. The Iterative and incremental methodology emphasizes design over documentation and is suitable for medium and large projects.

Spiral is a methodology that focuses on identifying objectives and analysing viable alternatives in the context well documented project constrains [4]. The Spiral methodology has 4 major phase: planning, risk analysis, development and evaluation. Project will fallow each phase multiple times in the above mentioned order until the software application is ready to be setup on the live environment. The Spiral methodology emphasizes risk analysis and always evaluates multiple alternatives before proceeding to implementing one.

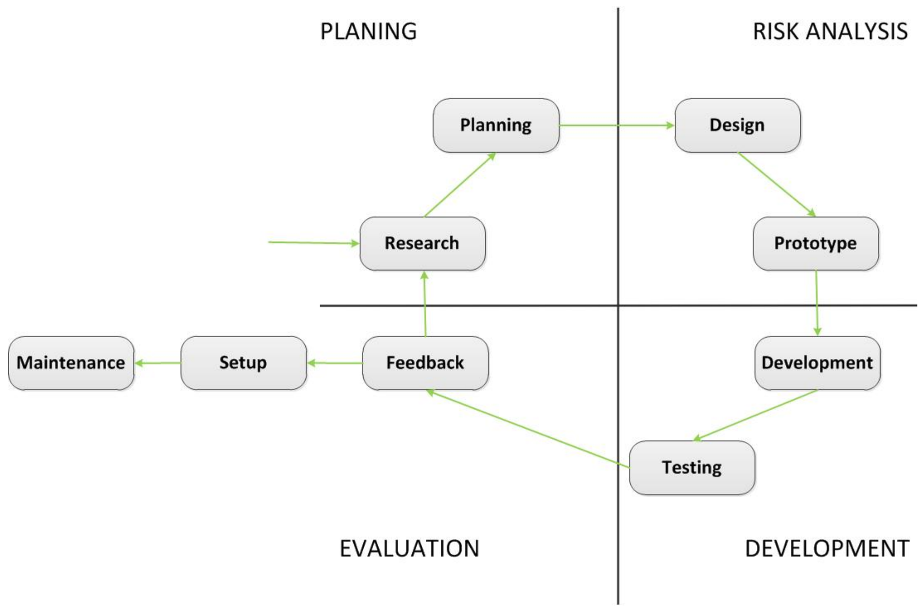


Fig. 4. Spiral methodology

The project owner's feedback is received after the first iteration of the spiral is completed. The Spiral methodology is suitable for medium and large scale projects. It has also proven more effective in implementing internal projects as identifying risks proprietary to your own organization is easier.

Rapid application development is a development lifecycle designed to give much faster development and higher-quality results than those achieved with the traditional methodologies. It is designed to take the maximum advantage of powerful development software [15]. Rapid application development imposes less emphasis on planning tasks and more emphasis on development. Development cycles are time boxed and multiple cycles can be developed at the same time.

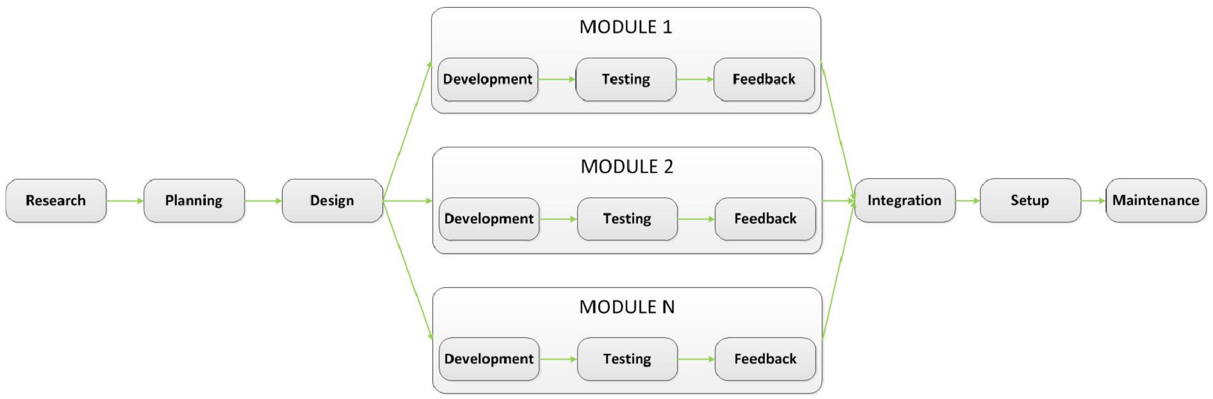


Fig. 5. Rapid application development methodology

The project owner's feedback is received after each module is completed. The Rapid application development methodology is suitable for small, medium and large scale projects with the constraint that projects have to be broken down into modules.

Extreme programming breaks the conventional software development process into smaller more manageable chunks. Rather than planning, analysing, and designing for the entire project at once, extreme programming exploits the reduction in the cost of changing software to do all of these activities a little at a time, throughout the entire software development process [5]. It enforces pair programming where two developers use the same computer. One is writing code and the other is supervising. They change roles at regular intervals. For reducing the number of errors it relies heavily on unit testing and developers are required to write tests before writing the actual code. There is a collective ownership code policy where any developer can change any code sequence even if it was not written by him. The project owner is the one that decides the priority of the tasks.

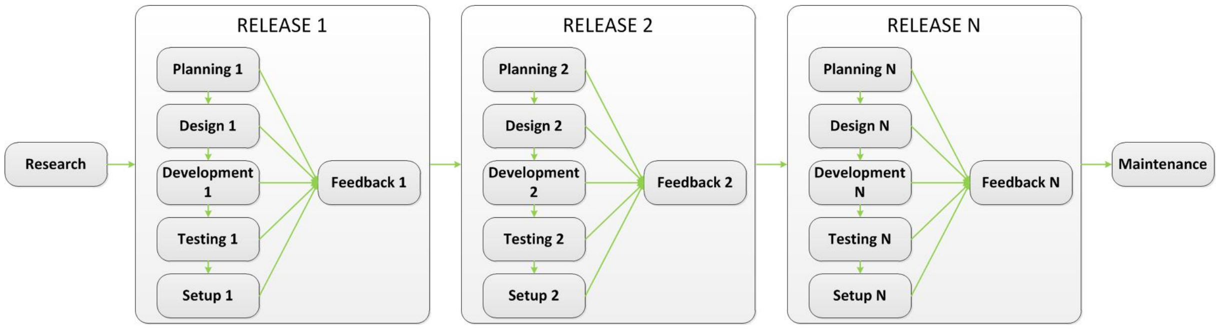


Fig. 6. Extreme programming methodology

Extreme programming methodology requires that a represented of the project owner is always with the development team in order to have access to continuous and relevant feedback. The Extreme programming methodology is suitable for small, medium and large scale projects.

V-Model methodology is a software development process which is an extension of the waterfall model [16]. It emphasizes thorough testing by pairing each software development stage with a matching phase of testing.

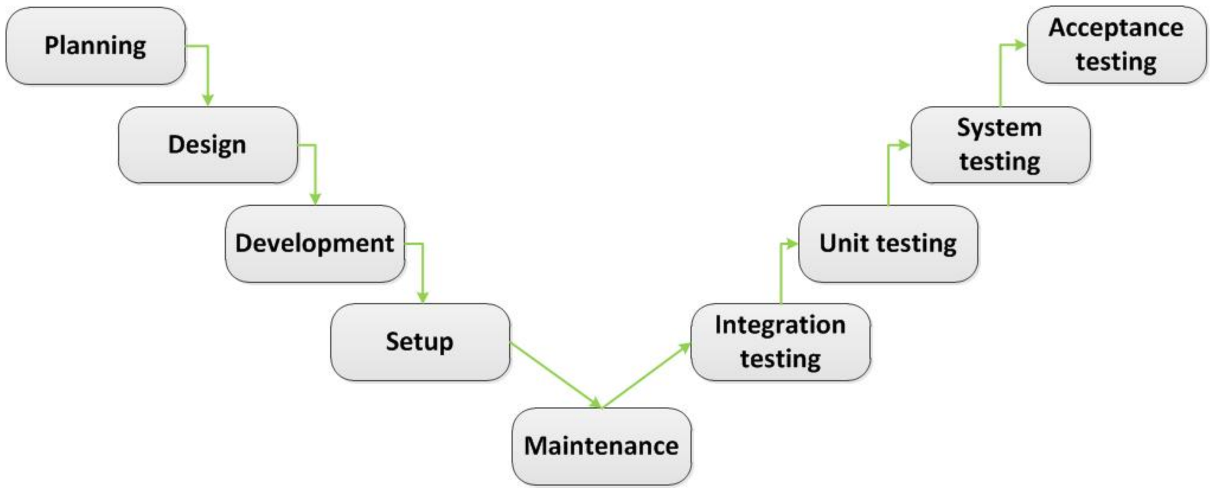


Fig. 7. V-Model methodology

The project owner's feedback is received in the form of acceptance testing after the entire application is completed. The V-Model development methodology is suitable for small and medium scale projects.

Scrum is a methodology for incrementally building software in complex environments [6]. Software requirements are formulated and prioritized by the product owner and are called stories. All the stories make up the Product Backlog. The Scrum methodology adopted a time box approach where development cycles known as Sprints take no more than 4 weeks and end with a working version of the application. All the stories of a sprint make up the Sprint's Backlog. Progress is assessed in daily meetings that are confined to 15 minutes and are known as Daily Scrum. Task assignment is not done by the project manager or by any other individual. Scrum development teams are self-organized and task assignment is a process where every team member is involved. The team efforts are kept on track by a Scrum Master.

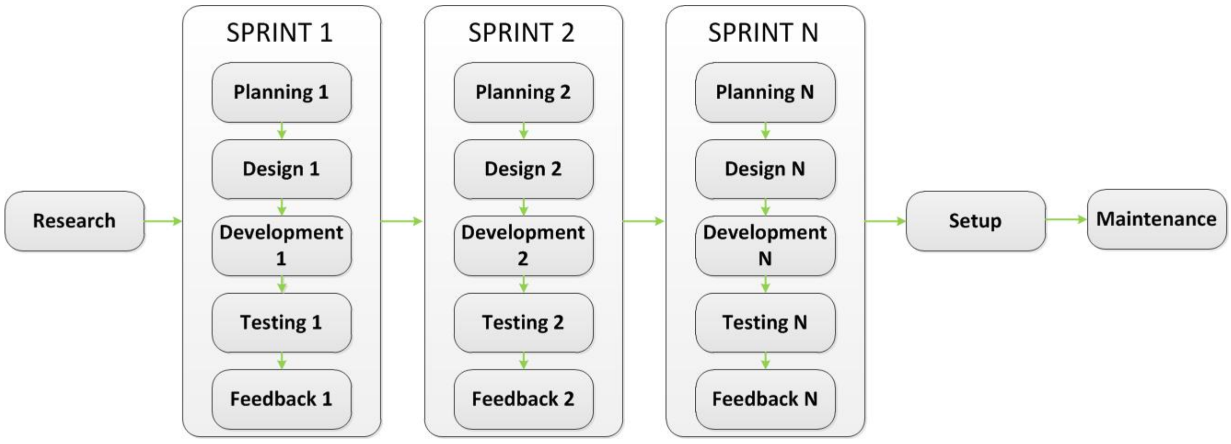


Fig. 8. Scrum methodology

The project owner's feedback is received at the end of each sprint. The Scrum methodology is suitable for small, medium and large scale projects.

Cleanroom is a methodology that focuses on defect prevention. The motivation for this approach is the fact that defect prevention is much less expensive than defect removal. The goal of the Cleanroom methodology is to construct software with no defects during development [7]. Cleanroom methodology relies on a box structure method to design the software product. Quality control is performed by using mathematic models and it also introduces a statistical approach to testing. Developers do not test the code that is the testing team's responsibility.

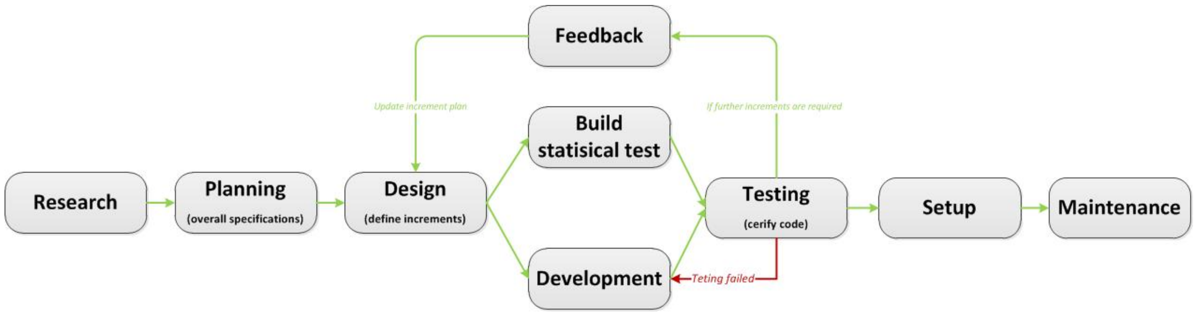


Fig. 9. Cleanroom methodology

The project owner's feedback is received at the end of each increment. The Cleanroom methodology is suitable for small, medium and large scale projects.

Dynamic systems development methodology is focused on developing application systems that truly serve the needs of the business [17]. Dynamic systems development methodology is an iterative development model that uses a timebox approach and MoSCoW task prioritization. It defines strict quality standards at the beginning of the project and it sets non-negotiable deadlines. Testing is done early and continually throughout the entire development cycle.

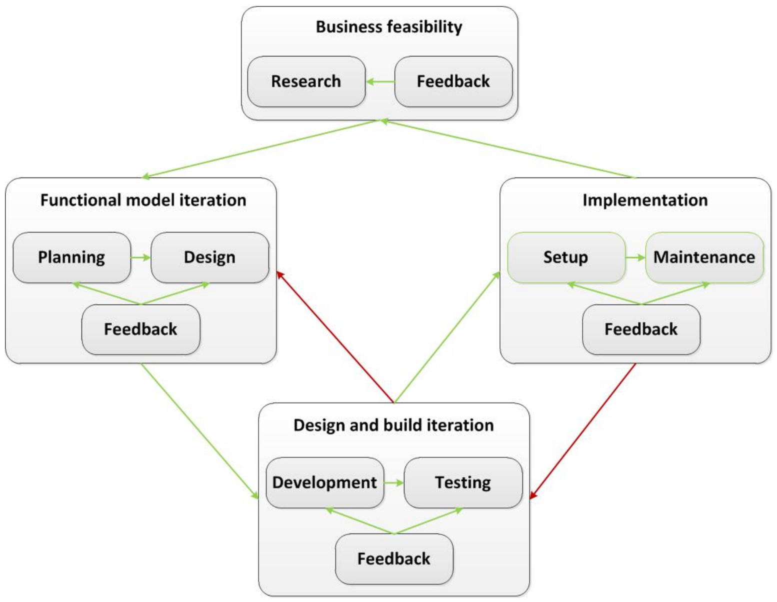


Fig. 10. Dynamic systems development methodology

Project team and project owner share a workplace, physical or virtual, in order to facilitate efficient feedback at every stage of the project. The Dynamic systems development methodology is suitable for medium and large scale projects.

Rational unified process methodology provides a disciplined approach to software development. It comes with several out-of-the-box roadmaps for different types of software projects and it provides guidance for all aspects of a software project. It does not require the project team to engage in any specific activity or produce any specific artefact. It provides guidelines that help the project manager tailor the process if none of the out-of-the-box roadmaps suits the project or organization [8].

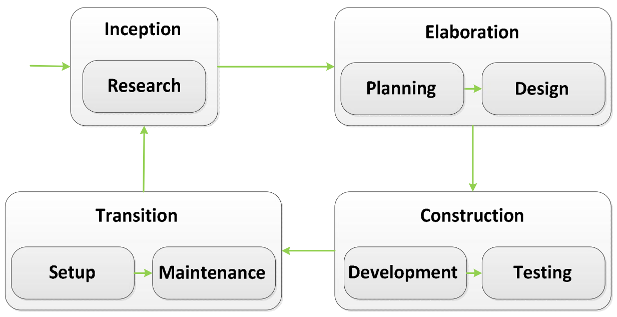


Fig. 11. Rational unified process methodology

The project owner's feedback is received as agreed at the start of the project as the methodology does not enforce a rule on project team – project owner collaboration. The Rational unified process methodology is suitable for small, medium and large scale projects.

Lean software development is a product development paradigm with an end-to-end focus on creating value for the project owner and for the end user, eliminating waste, optimizing value streams, empowering people and continuously improving [9]. Value is defined as something that the project owner would pay for. Anything that is not adding value is considered waste and has to be discarded. Lean software development delivers early iterations of working code. It also insures that team members are motivated by empowering them to make significant decisions regarding the application. The Lean software development methodology does not enforce a certain process in terms of conducting the project. Project manager and team members are free to use any process they see fit as long as it stays true to core lean development principals.

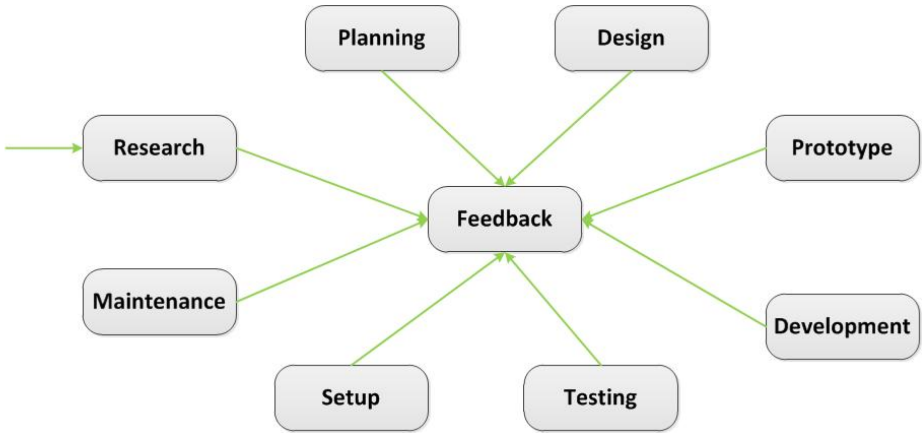


Fig. 12. Lean software development methodology

The project owner's feedback is central to the Lean software development methodology. The Lean software development methodology is suitable for small, medium and large scale projects.

Test-driven development is a methodology developed around unit testing. Before writing any actual code the developers write automated test cases for new functionality. If the tests work then there is no need to write any code as the functionality already exists, it was just not know to the developer. This scenario is often encountered when dealing with legacy code. If tests do not compile then the developer will write the code and run the tests again. The process is repeated until all requirements are met [10].

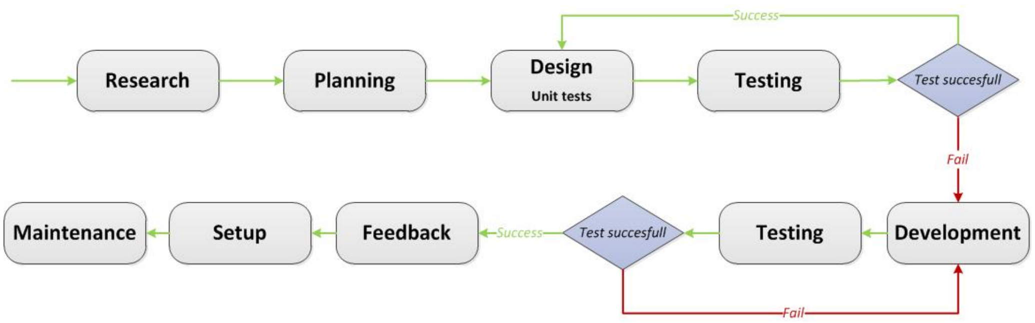


Fig. 13. Test-driven development methodology

The project owner's feedback is received after the code tested successfully. The Test-driven development methodology is suitable for medium and large scale projects. It is also recommended for scenarios where developers have to work with legacy code.

Behaviour-driven development methodology is developed around acceptance testing. The project owner writes the requirements in the form of acceptance tests using a standard format. Requirements are defined as user stories and include a title, a narrative part and acceptance criteria [18]. Based on the acceptance test scenarios developers will implement functionality. When functionality in developed it is tested using the same acceptance testing scenarios. If it passes the tests code is moved on the live environment. The entire process is repeated until all requirements are met.

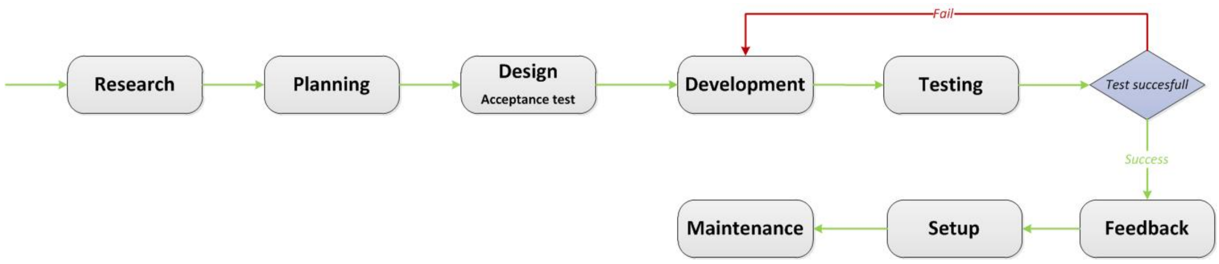


Fig. 14. Behaviour-driven development methodology

The project owner's feedback is received after the code tested successfully. The Behaviour-driven development methodology is suitable for small, medium and large scale projects. It is also recommended for scenarios where developers have to work with legacy code.

Feature-driven development is a methodology focused on actual functionality. Each feature in the Feature-driven development methodology reads as a requirement which is understandable by the project owner, it has true business meaning and describes true business value [11].

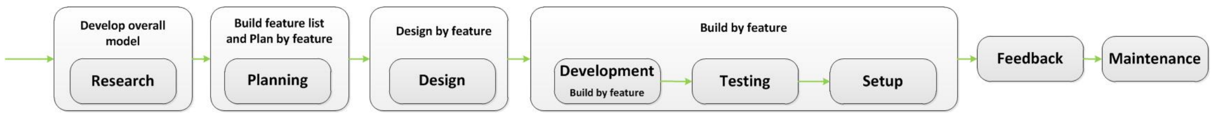


Fig. 15. Feature-driven development methodology

The project owner's feedback is received after the application is already setup but constant interaction between development team and project owner takes place throughout the entire length of the project. The Feature-driven development methodology is suitable for small, medium and large scale projects.

Model-driven engineering is a complex methodology that uses domain models as ways of handling requirements [12]. Based on project owner's requirements a metamodel is define. The metamodel is actually a platform independent model that can be migrated onto any environment. UML is usually used for building the metamodel. The metamodel is then converted into a model that is specific to a certain platform. Based on the model the actual code is then generated.

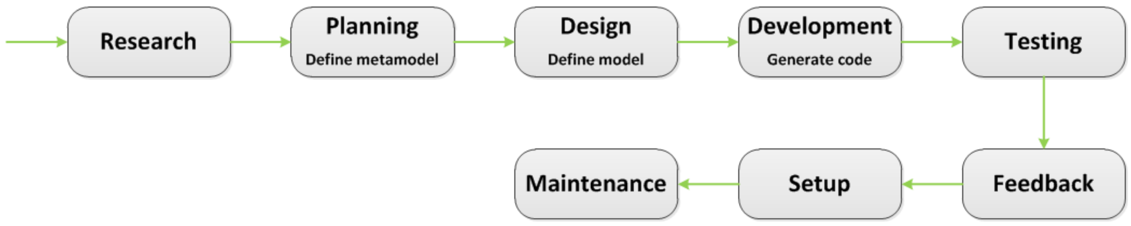


Fig. 16. Model-driven development methodology

The project owner's feedback is received after the code tested successfully. The

Model-driven engineering methodology is suitable for small, medium and large scale projects. It is also recommended for projects that will have long exploitation period as metamodels can be easily migrated and adapted to new technologies.

Crystal Methods is a family of methodologies that developed around the theory that people, and not tools or process, are the most important factor in any software project. Crystal Methods is a myriad of methodology elements and does not tackle every project in the same manner but instead uses custom tailored processes and tools depending on the project's profile and scale. Large or safety critical projects require more methodology elements than small non-critical projects. With Crystal Methods, organizations only develop and use as much methodology as their business needs demand [19]. Crystal uses an iterative approach but does not enforce a release with every iteration.

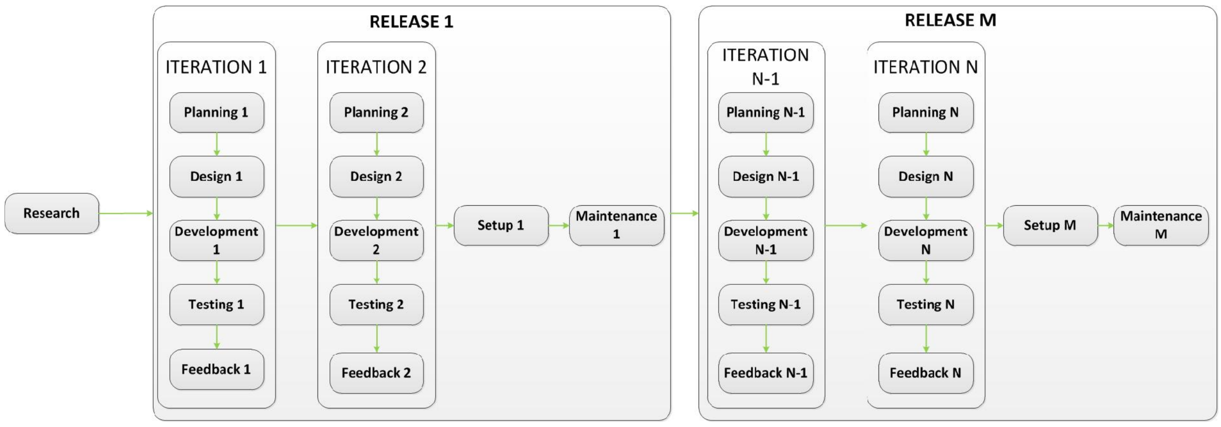


Fig. 17. Crystal Methods methodology

The project owner's feedback is received after each iteration is finished. The Crystal Methods methodology is suitable for small, medium and large scale projects. It has a different approach depending on the scale of the project.

Joint application development is a methodology that focuses on system requirement determination by involving end users, project owner and project team in a series of freely interacting meetings [13]. End user and project owner are also heavily involved in the design and development stages. Joint Application Development methodology uses prototyping as the basis for the actual software development.

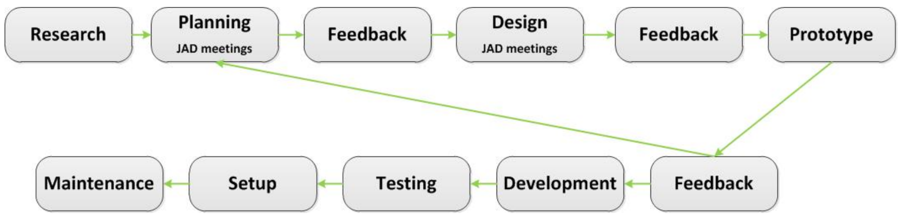


Fig. 18. Joint application development methodology

The project owner's feedback is received at every JAD meeting and after the prototype is finished. The Joint application development methodology is suitable for medium and large scale projects.

Adaptive software development is a methodology that was built as a response to an economy that is increasingly changing and evolving [14]. Adaptive software development is based on iterative development and is oriented on the project's mission. It is a timeboxed model that values delivering features and accepts changes in all stages of the project. Adaptive software development responds accepts risks and handles them efficiently.

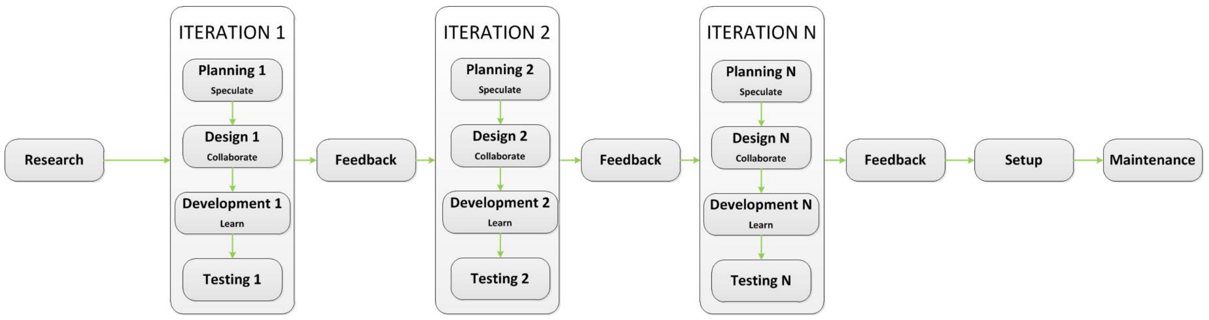
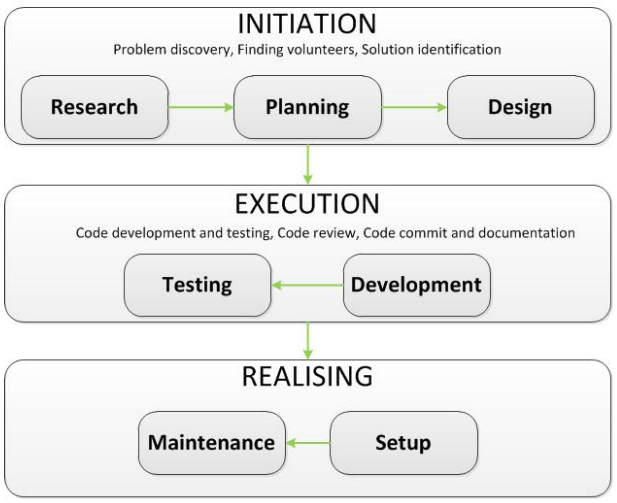


Fig. 19. Adaptive software development methodology

The project owner's feedback is received after each iteration is finished. The Adaptive software development methodology is suitable for small, medium and large scale projects.

Open source software development is a decentralized methodology with no central authority, project owner, no compensation for the project team, no accountability and yet with a high success rate [20]. Open source software development defies traditional economic theory as thousands of programmers work on writing, debugging and testing software without expecting any direct compensation. Most open source software developers will never meet face to face and yet find a way to collaborate in harmony. Open source software development has the advantage of comprehensive testing as code is reviewed by a large number of developers and also benefits from around the clock work on the project as developers are geographically scattered all around the globe.


INITIATION
Problem discovery, Finding volunteers, Solution identification
Research
Planning
Design
EXECUTION
Code development and testing, Code review, Code commit and documentation
Testing
Development
REALISING
Maintenance
Setup
Fig. 20. Open source software development methodology

In open source software development there is no project owner to provide feedback. The methodology is suitable for small, medium and large scale projects.

Microsoft Solutions Framework is a deliberate and disciplined approach to technology projects based on a defined set of principles, models, disciplines, concepts, guidelines, and proven practices from Microsoft [21]. Microsoft Solutions Framework methodology has versions for both lightweight and heavyweight implementation so it can be applied in an agile manner or using a waterfall approach. It fosters open communication and empowers team members but at the same time establishes clear accountability and shared responsibility.

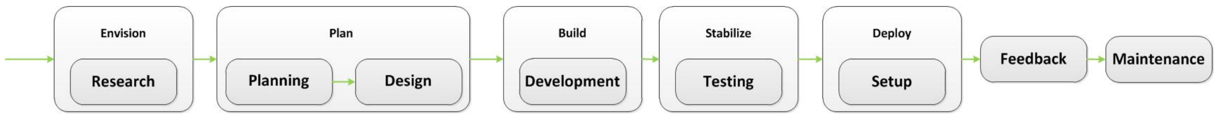


Fig. 21. Microsoft Solutions Framework methodology

The project owner's feedback is received after deployment. The Microsoft Solutions Framework is suitable for small, medium and large scale projects.

Apart from the above mentioned methodologies a further research of the topic might consider analysing the following methodologies:

- Dual Vee model – a variation of the V model;
- Evolutionary project management – forerunner of the current agile methodologies [23];
- Agile Unified Process – a simplified version of the Rational Unified Process;
- Essential Unified Process – a variation of the Rational Unified Process;
- Open Unified Process - a variation of the Rational Unified Process [22].

They were not included in the current comparative study as they are variation of other methodologies and their characteristics were already submitted to analysis.

4. Strengths and weaknesses

Software development methodologies follow one of two paths: heavyweight or lightweight. Heavyweight methodologies are derived from the waterfall model and emphasizes detailed planning, exhaustive specifications and detailed application design. Lightweight methodologies are derived from the Agile model and promote working software, individuals and interactions, acceptance of changing requirements and user feedback. The Microsoft Solutions Framework includes variations for both heavyweight and lightweight philosophies.

Table 2. Software development methodologies characteristics

Methodology	Characteristics	Strengths	Weaknesses
Waterfall	- comprehensive documentation - meticulous planning - linear-sequential process - each phase has its own deliverables	- easy to manage - easy to understand for the project owner and team	- working code is delivered late in the project - does not cope well with changing requirements - low tolerance for design and planning errors
Prototyping	- build one or more demo versions of the software product - project owner is actively involved - prototypes are meant to be discarded	- accurate identification of application requirements - early feedback from the project owner - improved user experience - early identification of missing or redundant	- leads to unnecessary increase of the application's complexity - increased programming effort - costs generated by building the prototype

	- writing code is valued over writing specifications	functionality	
Iterative and incremental	- build an initial model that is meant to be extended in successive iterations - emphasizes design over documentation - project owner is actively involved	- continuous feedback from the project owner - multiple revisions on the entire application and on specific functionality - working code is delivered early in the project	-each iteration is a rigid structure that resembles a small scale waterfall project
Spiral	- focuses on objectives, alternatives and constrains - has 4 major phase: planning, risk analysis, development and evaluation - emphasises risk analysis - evaluates multiple alternatives before proceeding to the planning stage	- working code is delivered early in the project - minimizes risk - strong documentation	- costs generated by risk handling - dependent on accurate risk analysis
Rapid application development	- less emphasis on planning tasks and more focus on development - timebox approach	- applications are developed fast - code can be easily reused	- poor documentation - high development costs - code integration issues - application has to be broken into modules
Extreme programming	- pair programming - unit testing - fast consecutive releases - collective ownership - on-site project owner - open workspace - project owner decides the priority of tasks	- application gets very fast in the production environment - frequent releases of working code - reduced number of bugs - smooth code integration - continuous feedback from the project owner	- lack of documentation - developers reluctance to pair programing - developers reluctance to write tests first and code later - requires frequent meetings - lack of commitment to a well-defined product leads to project owner reluctance
V-Model	- introduces testing at every development stage - highlights the importance of maintenance	- low bug rate -easy to understand and use	- vulnerable to scope creep - relies heavily on the initial set of specifications
Scrum	- iterative development - timebox approach known as Sprints - daily meetings to assess progress known as Daily Scrum - self organizing development team - tasks are managed using backlogs; product backlog and sprint backlog	- deliver products in short cycles - enables fast feedback - rapid adaptation to change	- lack of documentation - requires experienced developers - hard to estimate at the beginning the overall effort required to implement large projects; thus cost estimates are not very precise
Cleanroom	- iterative development - box structure method - using mathematic models in quality control - statistical approach to testing	- considerable reduction in bug rate - higher quality software products	- increased development costs - increased time to market for software product - requires highly skilled highly experienced developers
Dynamic systems development	- iterative development - MoSCoW prioritisation of	- focusses on addressing effectively the business	- requires large project teams at it has multiple

method	tasks	needs	roles to cover
	- timebox approach - non-negotiable deadlines - strict quality standards set at the beginning of the project - project team and project owner share a workplace (physical or virtual) - test early and continually	- post project implementation performance assessment - complete documentation - active user involvement	- requires very skilled developers
Rational Unified Process	- iterative development - prioritize risk handling - adequate business modelling - change management - performance testing	- accurate and comprehensive documentation - efficient change request management - efficient integration of new code - enables reuse of code and software components	- requires highly qualified professionals - development process is complex and poorly organized
Lean software development	- iterative development - discards all components that do not add value to the product - amplify learning - customer focus - team empowerment - continuous improvement	- reduced project time and cost by eliminating waste - early delivery of working code - motivated project team	- project is highly dependable on individual team members - a team member with strong business analysis skills required
Test-driven development	- unit testing -testing scenarios are developed before actual coding - repeated short development cycles - suitable for debugging legacy code developed with other techniques	- less time spent on debugging - higher quality code - by designing tests the developer empathizes with the user - less defects get to the end user	- tests are focused on syntax and overlook actual functionality - requires more code than most methodologies - the developer is actually the one doing the testing - writing unit tests increases costs
Behavior-driven development	- unit testing - focuses on business value - genuine collaboration between business and development	- easy to maintain - usability issues are discovered early - reduced defect rate - easy to integrate new code	- project owners are reluctant to write behaviour scenarios
Feature-driven development	- iterative development - application is broken down into features - no feature should take longer than two weeks to implement - uses milestones to evaluate progress	- multiple teams can work simultaneously on the project - scales well to large teams - good progress tracking and reporting capabilities - easy to understand and adopt	- individual code ownership - iterations are not well defined
Model-driven engineering	- uses domain model - models are automatically transformed into working code - knowledge is encapsulated in high level models - emphasizes reuse of standardized models	- high degree of abstraction - increased productivity - delivers products with a high degree of compatibility and portability - shorter time to market	- requires considerable technical expertise - documentation is readable only by domain experts - is difficult to implement version control on modelling environment

		- lowers maintenance costs	
Crystal Methods Methodology	- focusses on people and skill and not on process - more than one iteration in a release - different approaches depending on the projects size and criticality	- easy to implement - frequent delivery of working code - developers have dedicated timeslots to reflect on possible code improvements	- critical decisions regarding the architecture of the application are made by individuals and not by the entire team
Joint Application Development	- emphasises system requirement determination - involves the project owner and end user in the design and development - JAD meetings - prototyping	- accelerates design - enhances quality - promotes teamwork with the customer - creates a design from the customer's perspective - lowers maintenance costs	- relies heavily on the success of the group meetings - does not have a documented approach for stages that follow system requirements determination and design
Adaptive software development	- iterative development - focusses on the final goal of the project - feature based - timeboxed - risk driven	- effective handling of change and scope creep - easy to understand and implement - enables innovation	- low risk handling - uses assumptions and predictions - lacks tangible documentation
Open source software development	- iterative development - geographically distributed teams - collaborative work	- low costs - highly motivated and dedicated developers - comprehensive testing as code is reviewed by a large number of developers	- low accountability for submitted code - no central management authority - unstructured approach to development
Microsoft Solutions Framework	- has versions for both lightweight and heavyweight implementation - fosters open communication - empower team members and establishes clear accountability and shared responsibility	- supports multiple process approaches - solid risk handling policies - built to respond effectively to change - reduces team size	- difficult to setup and configure

Out of a total of 20 software development methodologies that were analysed 6 were based on the heavyweight model and 13 were based on the lightweight model. One methodology offered support for both a lightweight and a heavyweight approach. In order to choose the appropriate software development methodology for a project one should consider: project owner profile, developer's technical expertise, project complexity, budget and deadlines. An innovative software development project is difficult to match with one of the existing development methodologies. Innovative software development projects require writing comprehensive documentation in order to patent any original output that might result. It also requires a considerable degree of flexibility as changes occur often.

5. Conclusions

Software development methodologies follow two major philosophies: heavyweight and lightweight. Heavyweight methodologies are suitable for projects where requirements are unlikely to change and the software complexity allows for detailed planning. Heavyweight methodologies are easy to understand and implement. They provide solid documentation and appeal to project owners because they are well structured and showcase tangible deliverables for every stage of the project. With heavyweight methodologies the project manager can easily perform tracking, evaluation and reporting. The project owner is considerably involved only in the research and planning stages. Lightweight methodologies are suitable for projects were specifications are unclear or are likely to change due to project internal or external factors. Lightweight methodologies are based on an incremental approach were software is delivered in multiple consecutive iterations, all of them being functional versions of the application. Lightweight methodologies provide great flexibility and can easily adapt to change. They promote early delivery of working code, self-organizing teams and adaptive planning. The project owner is heavily involved in all the stages of the project as its input and feedback is critical for the success of lightweight methodologies. When choosing a software development methodology project owner profile, developer's technical expertise, project complexity, budget and deadlines must be taken into account. Often no methodology will fit perfectly the profile of a specific project. Then the best matching methodology should be used or in the case of experienced project teams and project managers a combination of methodologies could be introduced. In the case of innovative software development projects a new methodology is required. This topic is a subject for further research in the software development field.

6 Acknowledgment
This paper was co-financed from the European Social Fund, through the Sectorial Operational Programme Human Resources Development 2007-2013, project number POSDRU/159/1.5/S/138907 "Excellence in scientific interdisciplinary research, doctoral and postdoctoral, in the economic, social and medical fields -EXCELIS", coordinator The Bucharest University of Economic Studies.

References

[1] http://php.net/ChangeLog-5.php
[2] J. E. Cooling, T. S. Hughes, "The emergence of rapid prototyping as a real-time software development tool", *Proceedings of the Second International Conference on Software Engineering for Real Time Systems*, 18-20 Sep. 1989, Cirencester, UK, Publisher: IET, 1989, pg. 60-64
[3] C. Larman, V. R. Basili, "Iterative and Incremental Development: A Brief History", *Computer*, vol. 36, no. 6, pg. 47-56, 2003, doi:10.1109/MC.2003.1204375
[4] B.W. Boehm, "A spiral model of software development and enhancement", *Computer*, vol. 21, no. 5, pg. 61-72, 1988, doi: 10.1109/2.59
[5] K. Beck, "Embracing change with extreme programming", *Computer*, vol. 32 , no.10, pg. 70 – 77, 1999, doi: 10.1109/2.796139
[6] L. Rising, N. S. Janoff, "The Scrum Software Development Process for Small Teams", *IEEE Software*, vol. 17, no. 4, pg. 26-32, 2000, doi:10.1109/52.854065
[7] A. Spangler, "Cleanroom software engineering-plan your work and work your plan in small increments", *IEEE Potentials*, vol.15, no. 4, pg. 29 – 32, 1996, doi: 10.1109/45.539962
[8] G. Pollice, *Using the Rational Unified Process for Small Projects: Expanding Upon eXtreme Programming*, Rational Software White Paper, 2001
[9] C. Ebert, P. Abrahamsson, N. Oza, "Lean Software Development", *IEEE Software*, vol. 29, no. 5, pg. 22-25, 2012,
[10] E. M. Maximilien, L. Williams, "Assessing test-driven development at IBM", *Proceedings. 25th International Conference on Software Engineering*, ICSE 2003, 3-10 May 2003, Portland, USA, Publisher: IEEE, 2003, doi: 10.1109/ICSE.2003.1201238, pg.564-569
[11] D. J. Anderson, *Feature-Driven Development*, Microsoft Corporation, Oct. 2004
[12] J. Bezivin, *Model Driven Engineering: An Emerging Technical*

Space, International Summer School, *Summer School on. Generative and Transformational Techniques. in Software Engineering*, 4-8 Jul. 2005, Braga, Portugal, Publisher: Springer Berlin Heidelberg, pg. 36-64, doi: 10.1007/11877028_2.

[13] E. W. Duggana, C. S. Thachenkaryb, "Integrating nominal group technique and joint application development for improved systems requirements determination", *Information & Management*, vol. 41, no. 4, pg. 399–411, 2004, DOI: 10.1016/S0378-7206(03)00080-6

[14] D. Riehle, "A comparison of the value systems of Adaptive Software Development and Extreme Programming: How methodologies may learn from each other", *Proceedings of the First International Conference on Extreme Programming and Flexible Processes in Software Engineering*, XP 2000, 21-23 Jun. 2000, Cagliari, Italy, pg. 35-50

[15] J. Martin, *Rapid application development*, Publisher: Macmillan Publishing, 1991, pg. 788, ISBN:0-02-376775-8

[16] S. Mathur, S. Malik, "Advancements in the V-Model", *International Journal of Computer Applications*, vol. 1, no. 12, pg. 29-34, 2010, doi: 10.5120/266-425

[17] J. Stapleton, P. Constable, *DSDM, dynamic systems development method: the method in practice*, Publisher: Cambridge University Press, 1997, pg. 192, ISBN-10: 0201178893

[18] M. Soeken, R. Wille, R. Drechsler, "Assisted behavior driven development using natural language processing", *Proceedings of the 50th International Conference on Objects, Models, Components, Patterns*, TOOLS 2012, 29-31 May 2012, Prague, Czech Republic, Publisher: Springer Berlin Heidelberg, 2012, doi: 10.1007/978-3-642-30561-0_19, pg. 269-287

[19] J. A. Livermore, "Factors that Impact Implementing an Agile Software Development Methodology", *Proceedings of IEEE SoutheastCon*, SECON 2007, 22-25 Mar. 2007, Richmond, USA, Publisher: IEEE, 2007, doi: 10.1109/SECON.2007.342860, pg.82-86

[20] G. Madey V. Freeh R. Tynan, "The open source software development phenomenon an analysis based on social network theory", *Proceedings of the 8th Americas Conference on Information Systems*, AMCIS, 9-11 Aug. 2002, Dallas, USA, 2002, pg. 1806-1813.

[21] G. Lory, D. Campbell, A. Robin, G. Simmons, P. Rytkonen, *Microsoft Solutions Framework version 3.0 Overview*, White Paper, June 2003.

[22] R. Balduino, *Introduction to OpenUP (Open Unified Process)* http://www.eclipse.org/epf/general/OpenUP.pdf

[23] S. Woodward, "Evolutionary project management", *Computer*, vol. 32, no. 10, pg. 49-57, 1999, doi: 10.1109/2.796109

[24] W. W. Royce, "Managing the development of large software systems: Concepts and techniques", *IEEE WESCON*, vol. 26, no. 8, pg. 1-9, 1970

Mihai Liviu DESPA has graduated the Faculty of Cybernetics, Statistics and Economic Informatics from the Bucharest Academy of Economic Studies in 2008. He has graduated a Master's Program in Project Management at the Faculty of Management from the Bucharest Academy of Economic Studies in 2010. He is a PHD Student at the Economic Informatics PHD School and he is currently Project Manager at GDM Webmedia SRL. His main field of interest is project management for software development.