DOI: 10.26565/2304-6201-2019-42-05

УДК 004.42

Application of Agile methodologies for software development

Holovko Dmytro, Vasylieva Larysa
V.N.Karazin Kharkiv National University, Svobody square 4, Kharkiv city, 61022, Ukraine
e-mail: ds.holovko@gmail.com

In this work evolution of software lifecycle models is explored. Firstly to lightweight and then to agile software development methodologies, and factors that have led to a search for ways to improve approaches to software development. Also "outdated" development designing approaches are compared with modern flexible and conclusions made whether the advantage of the latter over the firsts is absolute and whether or not the modern and only the modern should be used or maybe older approaches still have their advantages and it is too early to exclude them.

***Keywords**: Agile, Scrum, Kanban, methodology, development, designing, agile methodologies, family of methodologies.*

В этой работе исследована эволюция моделей жизненного цикла программного обеспечения до сначала облегченных, а затем и гибких методологий разработки программного обеспечения, а также факторы, которые привели к поиску путей улучшения подходов к разработке программного обеспечения. Также сравнивались «устаревшие» подходы к организации разработки с современными гибкими, и сделаны выводы о том, является ли преимущество последних перед первыми абсолютным, и действительно ли они и только они должны использоваться на практике, или, возможно, более старые подходы все еще имеют свои преимущества, и еще слишком рано их исключать.

***Ключевые слова:** Agile, Scrum, Kanban, методология, разработка, проектирование, гибкие методологии, семейство методологий.*

В даній роботі досліджено моделі життєвого циклу та сімейство Agile методологій, наведено цінності та принципи маніфесту Agile та пояснено значення перших, наведено закономірності, загальні для всіх наведених моделей життєвого циклу, наведено власне моделі життєвого циклу та пояснено сутність кожної моделі, досліджено еволюцію моделей життєвого циклу програмного забезпечення до спочатку легких і потім гнучких методів розробки програмних продуктів, наведено ці гнучкі методи розробки і дано коротку характеристику кожної, та досліджено чинники, що призвели до пошуку шляхів вдосконалення підходів до розробки програмного забезпечення. Тобто, освітлюються ті проблеми, які намагались вирішити досвідчені спеціалісти, з якою метою вони створювали маніфест Agile, і намагались винайти спосіб розробки, що має на увазі постійно мінливі ринок, вимоги замовника та бажання користувачів без втрат ефективності, часу, та фінансів. Ця тема є дуже актуальною, тому, як Agile-методології є доволі новим підходом до розробки програмних продуктів, навіть в такій загалом новій науці, як комп'ютерні технології, особливо в Україні, де з'являються останні 5-7 років, і досі відбувається процес їх впровадження, тому як часто їх використовують лише «номінально», тобто із порушеннями всіх правил методології та Agile загалом, або навпаки бездоганно виконують всі правила та ритуали, що призводить до зменшення ефективності роботи команди та створює зайвий стрес, а західні підприємства використовують їх дещо більше десятиліття, хоча ідеї для деяких було знайдено у підходах до організації виробництва ще 60-х років 20-го сторіччя. Також порівнено «застарілі» підходи до організації розробки із новими гнучкими, пояснено в чому полягає їхня перевага та наведено висновок чи є ця перевага останніх над першими абсолютною і чи лише Agile методології мають бути використовувані на практиці, або ж і старіші підходи все ще мають свої переваги і їх ще зарано виключати.

***Ключові слова:** Agile, Scrum, Kanban, методологія, розробка, проектування, гнучкі методології, сімейство методологій.*

Waterfall

The waterfall lifecycle model is probably the first to be met by a person who starts to study the development methodology. The essence of the model is that the work on the project consists of a rigid sequence of stages: definition of functional and non-functional requirements, analysis, design, implementation, testing, integration and support (Figure 1.1), and the next stage cannot begin, until the previous one is completed [1]. This is precisely a problem with the waterfall model: if, for example, significant errors are detected during the testing, the project redesign will be needed and the whole project will go back through design, implementation and testing phases [1]. Therefore, the main drawbacks of the waterfall model are discrepancy with the real software creation conditions, lack of the possibility to return to the previous stages of development, error correction performed only during the testing phase and the project result could be obtained only at the end of the development process. There are also the advantages, however: the waterfall model is understandable, therefore it makes it easy to manage a project and allows you to determine in advance the cost and time estimations of development, it has a certain sequence of stages and allows you to evaluate the quality of the product on each of them, and with a clear definition of requirements and means of their implementation let you get excellent result quite quickly [2].

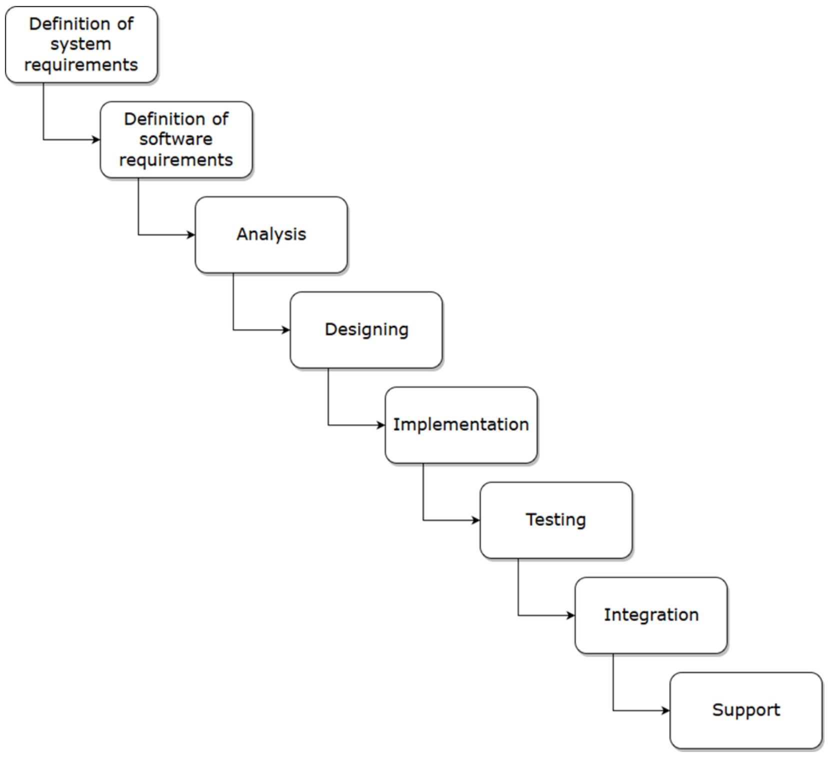

Definition of system requirements
Definition of software requirements
Analysis
Designing
Implementation
Testing
Integration
Support

Figure 1.1. Waterfall lifecycle model

Waterfall model with intermediate monitoring

In order to eliminate the main disadvantage of the waterfall model, it has been improved by adding backward links between stages. That is, an opportunity to go back to the previous stage (Figure 1.2) has been added, which allows solving problems at each stage immediately after the problem is detected, and to return to the previous steps where an error has been made. However, such a model has showed a 10-fold increase in development costs [2]. It is caused by the fact that because of returning to previous stages the cost proved to be stretched over time [3].

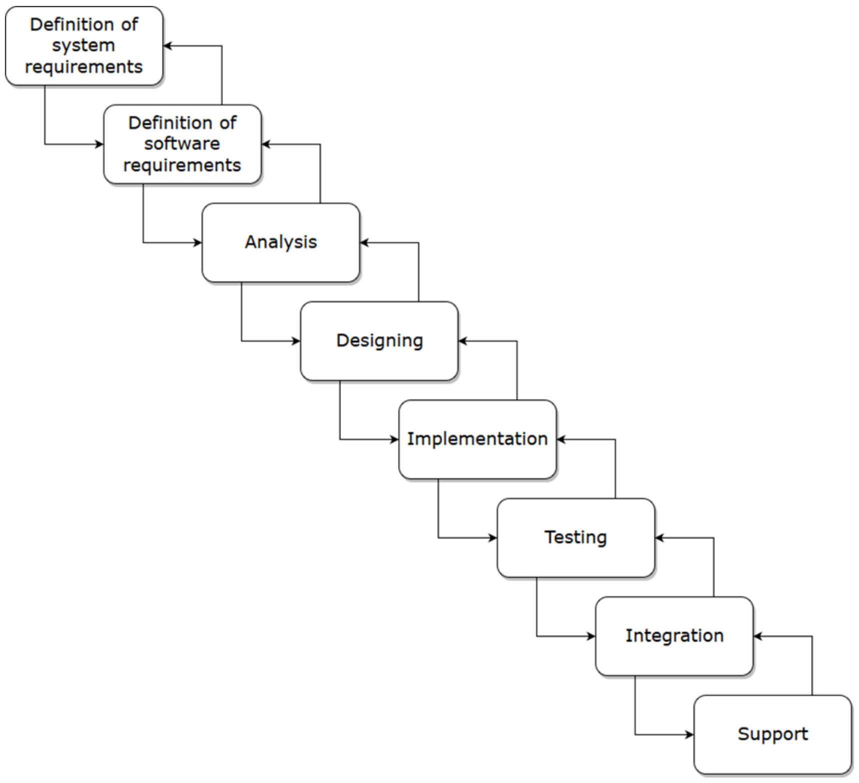

Definition of system requirements
Definition of software requirements
Analysis
Designing
Implementation
Testing
Integration
Support

Figure 1.2. Waterfall lifecycle model with intermediate monitoring

V-model of software lifecycle

Development through testing (V-model) - a model created in attempt to find a program product development algorithm that is close to real-life conditions [2]. It has a phased structure, as well as a waterfall model, but pays a lot of attention to verification and testing of the product being done at the same time as the development phase (Fig. 1.3), therefore this model is optimal for systems where a smooth operation is particularly important [4]. The V-model had some disadvantages however; the main one is late testing, which makes it impossible to make changes without changing the schedule of the project.

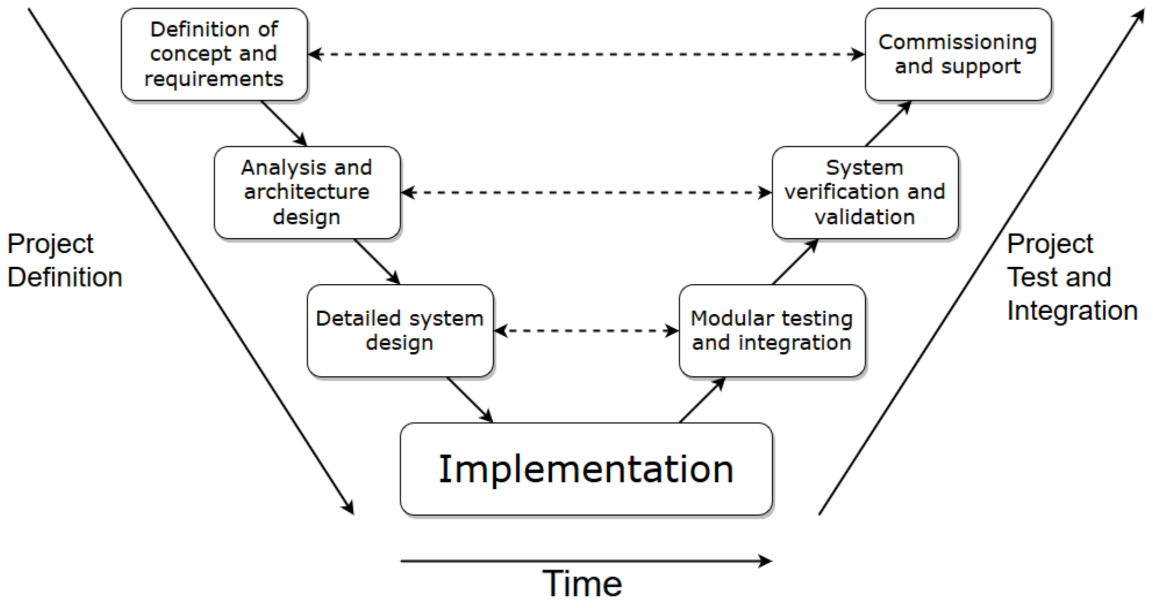

Definition of concept and requirements
Commissioning and support
Analysis and architecture design
System verification and validation
Project Definition
Project Test and Integration
Detailed system design
Modular testing and integration
Implementation
Time

Figure 1.3. V-model of the software lifecycle

General features of lifecycle models

Lifecycle models similar to any described above reflect the same states, from the moment the need in this software product arises and to the moment of its complete decommissioning. The indicated models differ in the interrelation of the lifecycle stages, but each of these stages is present in each model in one form or another [3].

Strategy definition – the name of this stage in different models usually contains such keywords as: definition, concept, requirements, project planning. That is, this stage, which can be presented in a particular model as several stages, usually a business survey and an assessment of the scope of the project, the goals and objectives, aims to do. At this stage, the entities and functions are determined at a high level [3].

This is done usually by a team of highly skilled business analysts having access to a leading staff of the customer company; main users of the future system usually participate in this phase, as well as domain experts [3].

The purpose of this phase, as noted above, is to obtain as much information as possible about the system, customer requirements and to transfer all this information in a formalized form to system analysts, and this is usually done through discussions with the customer, experts and users [3].

The result of this stage (strategy determining) is the documentation, where it is clearly stated what the customer will get if they agree to finance the project, time and costs estimations for the finished product [3]. The document should contain not only the costs but also the possible profit for the customer; for example the payback time of the project, the expected economic effect in cases where it can be estimated [3].

The **analysis** phase involves a detailed study of the business processes and the information needed for their implementation [3]. All information about the system, collected at the strategy definition stage is being formalized and specified at the stage of analysis [3]. Particular attention is paid to the completeness of transferred information, information analysis for non-contradictoriness, as well as the search for unused or duplicated information, because as a rule, the customer initially forms requirements not for system as a whole, but for separate components [3]. At this stage, the necessary components of the testing plan can also be determined [3].

Analysts collect and fix information in two related forms: functions and entities [3]. That is why results of the analysis stage traditionally a hierarchy of functions that divides the process of processing into components (what is done and what is it composed of), and the entity relationship model, which describes the entities, their attributes, and the relationships between them [3].

The **design** stage in its original historical form has been designed to create schematics of databases or data repositories, if any present in the project, and a set of specifications of future system modules, based on the data obtained from analysis phase. Also historically, a testing plan is finalized at the design stage, and a final document, a technical specification, according to which software will be developed, is created. As general, in mentioned above software lifecycle models, the design phase performs the following tasks:

1. complete verification of analysis stage's results;
2. determination of system constraints;
3. determination of the system's architecture;
4. determination of development tools to be used;
5. repository or database designing;
6. designing the interaction interfaces;
7. determination of specifications for system modules;
8. determination of testing requirements;
9. determination of system security requirements [3].

An important moment - all specifications should be as accurate as possible: those obtained after the stage of analysis as well as developed at this stage [3]. Quality of the design defines quality of the future product.

During the project **implementation** developers receive a specification and begin to write a code of the modules, strictly adhering to the designers' documentation [3]. The result of this stage is a program that is passed to the **testing** stage, which, in general, determines and evaluates quality of the system and its readiness for commissioning. That is why these two stages are not combined or parallel only in the waterfall model - as already mentioned, from the testing stage, it should return to the stage of implementation at least. In other models, this disadvantage has been eliminated in one way or another; actually attempts to eliminate the need to go back to the previous stages of testing have led to the emergence of such a number of software lifecycle models. In practice, the development and testing processes are running simultaneously, and developers are constantly cooperating with testers in order to eliminate most of the errors efficiently and in time. Sometimes these collaborations involve software designers who act as experts responding to developer requests for technical specifications.

The result of the testing phase is the system resistant to failures, malfunctions and capable to restore itself correctly [3].

The stage of **implementation or integration** involves the process of transition to a phase of operation. The system can be deployed completely, or gradually, depending on the model used, but in any case at this stage the system passes the so-called "user acceptance test" - the system begins to work "for real" [3].

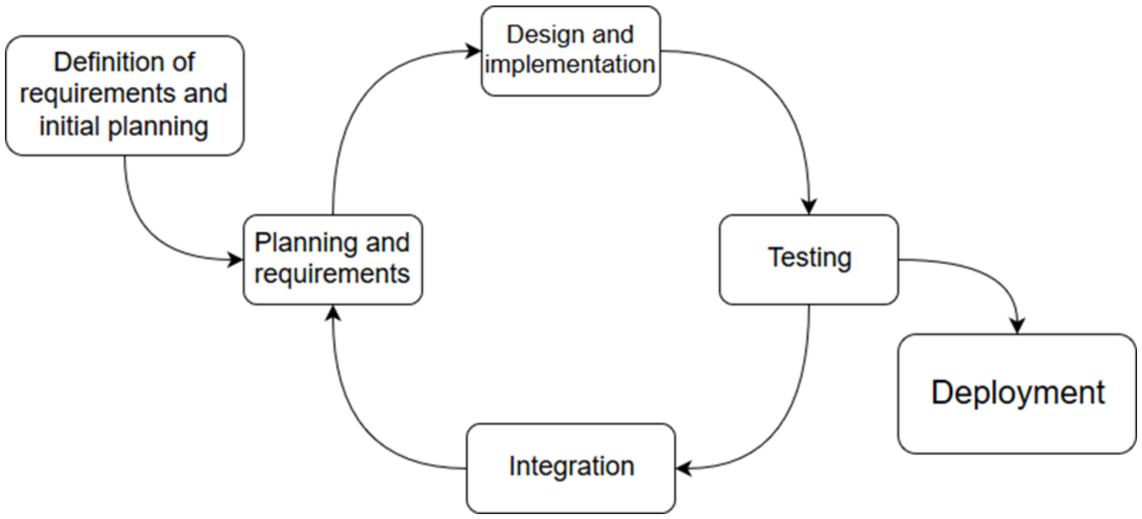
The stage of **operation and technical support** (if present) is the last stage of the software lifecycle. It can be noted that if technical support is available, the terms of support, the necessary staff and equipment are discussed with the customer and fixed in a separate document [3].

Incremental lifecycle model

A significant step towards Agile methodologies was made by separating the entire product development process into several development cycles, each cycle passing through the same phases (Figure 1.4). Such decomposition instead of a large inert project, which then needs to be tested as a whole, presents some small easy-to-create modules that are much easier to test and fix. Complete system requirements are also divided according to modules that will be developed per cycle.

According to the incremental model, in the first large loop the product is released with the basic functionality, and then new functions are gradually added, that means, that at each new so-called "increment" a new extended and improved version of product is being developed, and this process continues until a complete system is created [4].

The main advantage of this model is the swift response to changes in customer requirements: if the customer wants to introduce an additional functionality - it will be designed, implemented and tested in the next cycle; and shortly after release of first version of the product the reaction of audience, expediency and development risks could be assessed.


Definition of requirements and initial planning
Design and implementation
Planning and requirements
Testing
Deployment
Integration
Figure 1.4. Incremental software development model

Nevertheless, requirements for the whole system still must be defined before the implementation of the first version of the product, which imposes limitations. Moreover, the customer's requests needs to be periodically processed, formalized and adjusted to users' feedback and changes in the market.

Iterative lifecycle model

The main difference between the iterative model and the incremental one is that the start of the project doesn't require a complete specification of project requirements; the first cycle develops a part of functionality, which becomes the basis for the further development and specification of the requirements [4]. The first version may not be ideal - it just has to work, but each subsequent version brings the project closer to the desired result, and the result of each cycle is the working version of the product [4].

Rapid application development

Attention should be paid to the rapid application development model, or just "rapid application development", hereinafter referred to as RAD. This is the incremental model and the main peculiarity of RAD is that modules, components, or functions are developed by several groups simultaneously (Figure 1.5).

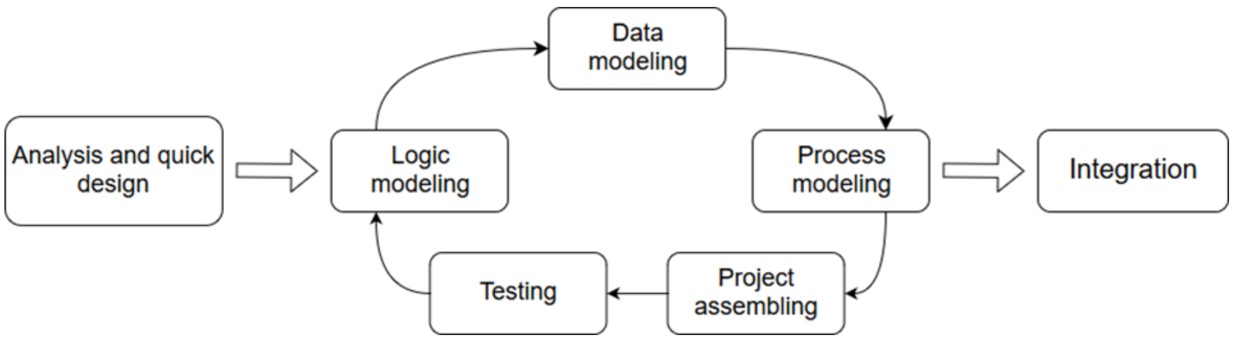


Figure 1.5. Rapid application development (RAD)

The time of one cycle is strictly limited, and upon completion, all created modules are combined into one working product [4]. The model cycle includes phases of "logic modelling", "data modelling", "process modelling", "project assembly" and "testing". In the first three phases the list of information flows, objects and entities, and relating objects is determined. RAD primarily uses automated assembling tools, which transform the generated models into a code at the fourth stage, the "project assembly". That is why the main limitation of the model is the high personnel requirements, highly skilled and specialized developers are required. In addition, the budget of such a project is usually rather high due to the need to pay for specialists' services and automatic assembling tools [4]. Therefore this model is used for "fast development" - it is effective when the project must be finished in a short time [4].

Spiral model

The spiral model that combines design and staged prototyping should also be mentioned [2].

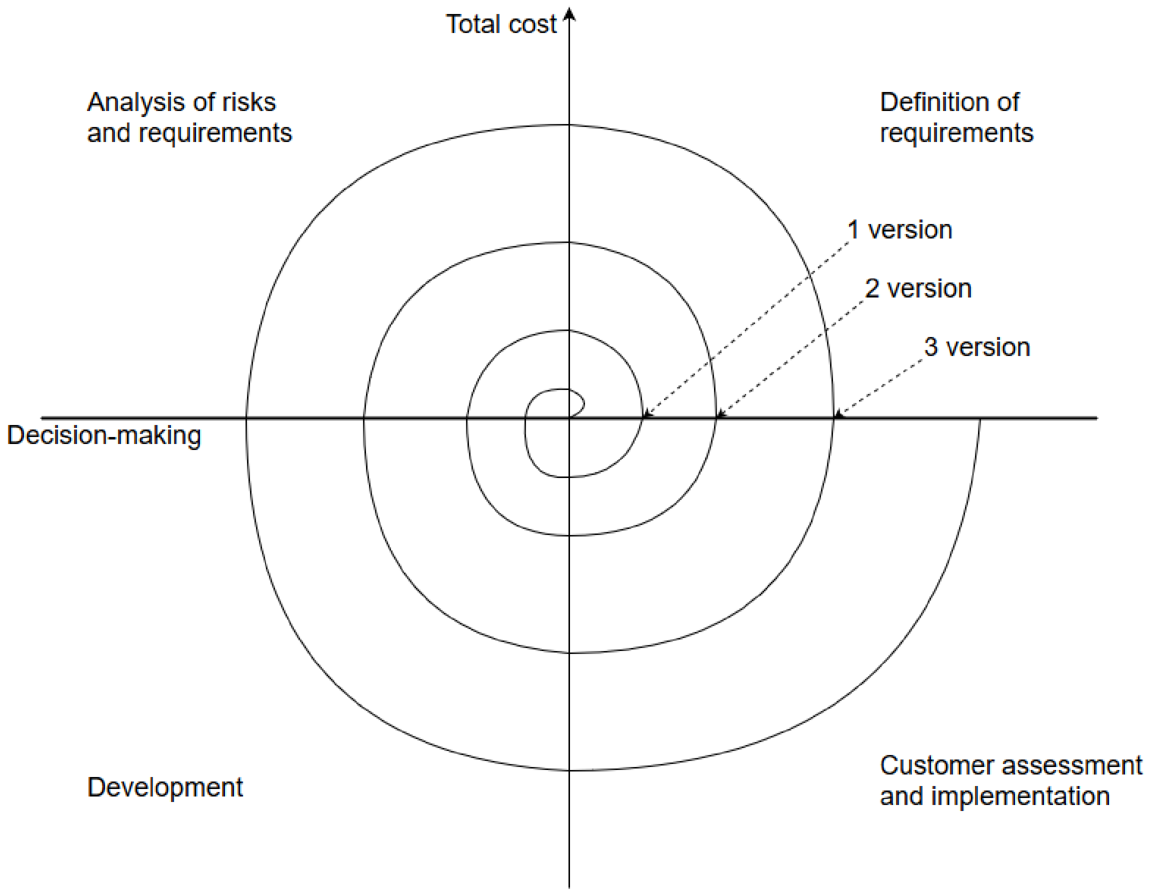

Total cost
Analysis of risks and requirements
Definition of requirements
1 version
2 version
3 version
Decision-making
Development
Customer assessment and implementation

Figure 1.6. Spiral software lifecycle model

Each "turn" of spiral consists of sequentially placed stages that have no backlinks and result of each "turn" is analyzed at the end of this "turn", the results of analysis affects the next "turn" (Figure 1.6). The "development" in the spiral model combines traditional stages of design, implementation and testing, that is, at this stage detection and correction of part of errors are performed. The errors which cannot be fixed and require more profound structural changes can be corrected on the next "turn" [4]. The spiral model, therefore, is very similar to the incremental one, but there is risk analysis present in each cycle. Therefore this model is suitable for the projects where each solution is critical and a failure is unacceptable [4]. That is why the efficiency of this model increases along with the size and the complexity of the project [3].

Historical factors of the Agile manifesto emergence, its values and principles

Now let us consider the methodology, because Agile is a methodology, not a model. The methodology is a system of principles, ideas, means, methods, and tools for software development, while the model is a description of software product lifecycle in stages with detailed explanation of each stage within the system [5]. In other words the methodology is a set of strategies for managing a product development process.

Agile is also called the "Agile methodologies family" which means that Agile is not a particular methodology, a clear manual or algorithm, and therefore allows for freedom of choice and adaptation, which is why a large number of methodologies exists under the common name of "Agile methodologies family", all of which are based on Agile principles, but have specific methods, practices and project management strategies [1].

Agile is a very young family of methodologies, its principles were formed in February 2001 by seventeen IT professionals gathered at a ski resort. The official website highlights this event as follows: "On February 11-13, 2001, at The Lodge at Snowbird ski resort in the Wasatch mountains of Utah, seventeen people met to talk, ski, relax, and try to find common ground—and of course, to eat" [6].

Shortly before this event, in the 1990s, a number of “light” software development methods were developed in response to the prevailing “heavy” methods that critics described as "overly regulated, planned, and micromanaged" [7]:

- Rapid application development (rapid application development, RAD) – 1991;
- Unified process (unified process, UP) – 1994;
- Dynamic systems development method (DSDM) – 1994;
- Scrum – 1995;
- Crystal Clear – 1996;
- Extreme Programming (XP) – 1996;
- Feature-driven development – 1997.

It is no surprise that the leading experts in the lightweight development methods were among the authors of the Agile Manifesto, which, in turn, gave impetus to the development of flexible methodologies and laid the foundations for a flexible approach to software development in general [7].

The Manifesto consists of four values and 12 principles and it contains no specific instructions.

1. Individuals and interactions over processes and tools;
2. Working software over comprehensive documentation;
3. Customer collaboration over contract negotiation;
4. Responding to change over following a plan [6].

Those definitions raise many questions due to their abstractness; therefore Scott Ambler's explanation can be useful:

1. Tools and processes are important, but more important that competent people work together effectively;
2. Good documentation is useful because it helps people understand how software is created and how to use it, but the main purpose of the development is to create software, not documentation;
3. The contract is important, but it is not a substitute for close cooperation with customers, which is needed to understand what they need;
4. The project plan is important, but it should not be too rigid; it should take into account changes in technology or the environment, stakeholder priorities, and people's understanding of the problem and its solution.

The twelve principles of Agile as follow:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software;
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage;
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale;
4. Business people and developers must work together daily throughout the project;
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done;
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation;
7. Working software is the primary measure of progress;
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely;
9. Continuous attention to technical excellence and good design enhances agility;

10. Simplicity - the art of maximizing the amount of work not done - is essential;
11. The best architectures, requirements, and designs emerge from self-organizing teams;
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly [6].

Nowadays there are a number of methodologies constituting Agile family and some of them have been mentioned before:

- Agile Modelling (AM) - this approach basically defines modelling procedures (including model code verification) and documentation within software development. The procedures for designing and constructing UML diagrams are described in a lesser extent and stages of development, testing, project management, deployment and maintenance are not affected.
- The Agile Unified Process (AUP) is a unified version of the RUP (IBM Rational Unified Process) methodology formulated by Scott Ambler. AUP defines a software development model for business applications.
- Agile Data Method (ADM) is a set of iterative tools of flexible software development with an emphasis on the formation of requirements and solutions through the cooperation of various cross-functional teams.
- Dynamic Systems Development Method (DSDM) is an iterative and incremental approach based on the Rapid Application Development (RAD) concept, which focuses on maximizing the end-user involvement in developing a software product and "Concentration on the useful 80% of the system, which can be implemented in 20% of the time".
- Essential Unified Process (EssUP) - an approach developed by Ivar Jacobson that contains methods of iterative software development with an emphasis on the product architecture and optimal team practices (essentially borrowed from RUP, CMMI and Agile Development). The idea is that only applicable in a particular situation practices and methods should be used. Based on the chosen methods and practices, the target process is determined. In contrast to RUP, where all practices and methods are interconnected, in this approach, there is flexibility and the ability to isolate exactly the necessary elements (methods and practices) from the available ones.
- Extreme programming (XP) - the idea of extreme programming is to use the best existing practices in the area of software development, raising them to a new (extreme) level. For example, in contrast to the usual practice, when one programmer consistently checks the written code after his colleague, in extreme programming, this check is performed in parallel. That increases the speed of product release, but also the risks too.
- Feature driven development (FDD) - the main idea of this approach is that “each function should be implemented in no more than two weeks”. Therefore, if it is unrealistic to develop a function as a whole, this function should be deconstructed into several functions and implemented gradually.
- Getting Real (GR) - this approach eliminates the functional specification procedures used for web applications. Development begins from the reverse, an interface and design are developed initially, and then the functionality itself.
- OpenUP (OUP) - this approach defines an iterative-incremental method of software development based on RUP. Within this method, the development lifecycle is defined (start-up phase, refinement phase, development and transfer phase to the customer). Due to certain stages and control points, the effectiveness of controlling and monitoring of the project's progress increases, which leads to the timely decision-making on the project.
- Lean software development - this approach is based on the concept of lean management of a manufacturing enterprise (lean production, lean manufacturing), the interaction of developers with customers and super-fast implementation of customer-critical functions.
- "Crystal" methodologies (Crystal Clear, Crystal Yellow and Crystal Orange, for example), the main points of which are choosing the policy, practical approaches and processes that will be valid for the entire project, and interacting with end-users.

And, of course, Scrum and Kanban, that are very common both in literature and real projects because for small projects and inexperienced teams they are quite optimal. Both methodologies are based on Agile principles and are designed for a small independent team of 5-9 people [8]. The team does not have an official head and decisions on the organization of the project are made without external intervention [8]. This results in collective responsibility - any error is a mistake of the entire team, not a specific member [8]. It is also highly recommended to place the team in one room, preferably without partitions. Contacts and communication are of most importance here and nothing should interfere.

Kanban

This method of the workflow organization comes from the automotive industry; the name itself is a Japanese term for the conveyor production method of Toyota in the '60s of the 20th century [9]. Its essence is to use so-called "main conveyor" or "main production" and "additional conveyors" or "additional productions", the main conveyor puts the pace of production, and additional cannot exceed this pace, but they can slow it down [9].

Kanban is based on three basic principles:

1. visualization of what is done today (the process of work) - the vision of all elements in the context of each other is the most informative;
2. limited work volume - this helps to balance the workflow as a flow of elements in development, and does not allow the team to start or run too much work at the same time;
3. rate support - when one job is completed, a new one, with the highest priority, is taken from the list for processing [10].

Therefore, Kanban is based on a prioritized list of tasks and progress stages of a specific task capacity, for example: "to processing", "in development", "in testing" and "implementation" (Figure 2.1).

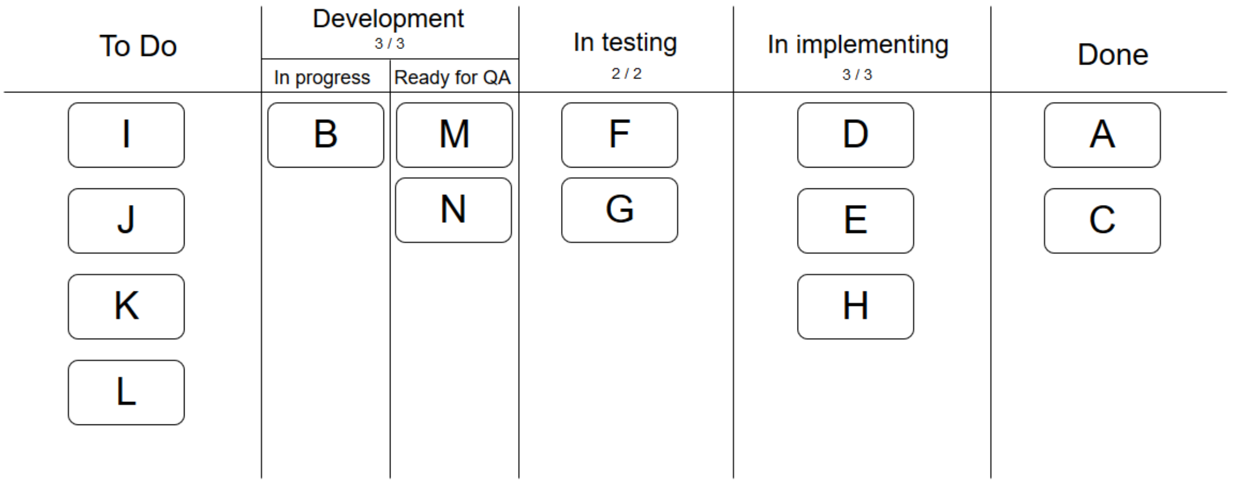


Figure 2.1. Example of workflow organisation with Kanban

This methodology is based on the priorities and developers' capabilities in terms of parallel work, rather than on time. When the developer is ready to take a new task, he takes it out of the list. When a new feature is enough to release a new version of the product – it is implemented on demand, and if there is an urgent important task – it is immediately taken to work. The capacity of each stage is limited by the total weight of tasks performed, and if a certain stage is divided into several, the capacity has a parental stage [8]. In the ideal case, all stages are filled to the maximum capacity, and in case of delays in one of stages, the developers from the least loaded stage can join temporarily (or permanently, if the delays at one and the same stage occur constantly) to eliminate delays throughout the product development process [9]. This liquidity is possible because teams in Agile are universal, and roles inside a team do not exist in Kanban [8].

The main purpose of Kanban - maximum speed of the passage of the task through all stages, which means the maximum development speed of new features of the product for the whole project, and this speed is more important than the number of simultaneously developed features, and Kanban itself is better suited for projects that have quickly and frequently changing priorities and requirements [10].

Scrum

Scrum has influenced the creation of the Agile Manifesto, but before it was the structure of projects, for which it was important to supply the new system features within more or less same time intervals, every two or four weeks, for example [10]. Therefore the first feature of Scrum is an iterative approach: all project work is divided into cycles of specific fixed duration, the so-called sprints (Figure 2.2).

The next feature is the two additional roles besides developers: a scrum-master and a product owner (PO) [8]. A scrum-master is a person whose duties are to organise the work, he is not a manager, does not hold a managerial position and does not gives instructions [8]. His direct duties are:

1. to hold a meetings;
2. to remove obstacles for teamwork;
3. to find and identify problems and to bring this information to the team;
4. to ensure that the team follows the methodology;
5. to monitor the progress of tasks [8].

In addition, the scrum-master is a member of the team and when not executing the duties above he works on the project as well [8].

The product owner defines the direction of the project development. It may be an external or internal client or a customer representative, but in any case, it is a person who has perfect knowledge of the market, the target audience and approves the results of the team's work [8].

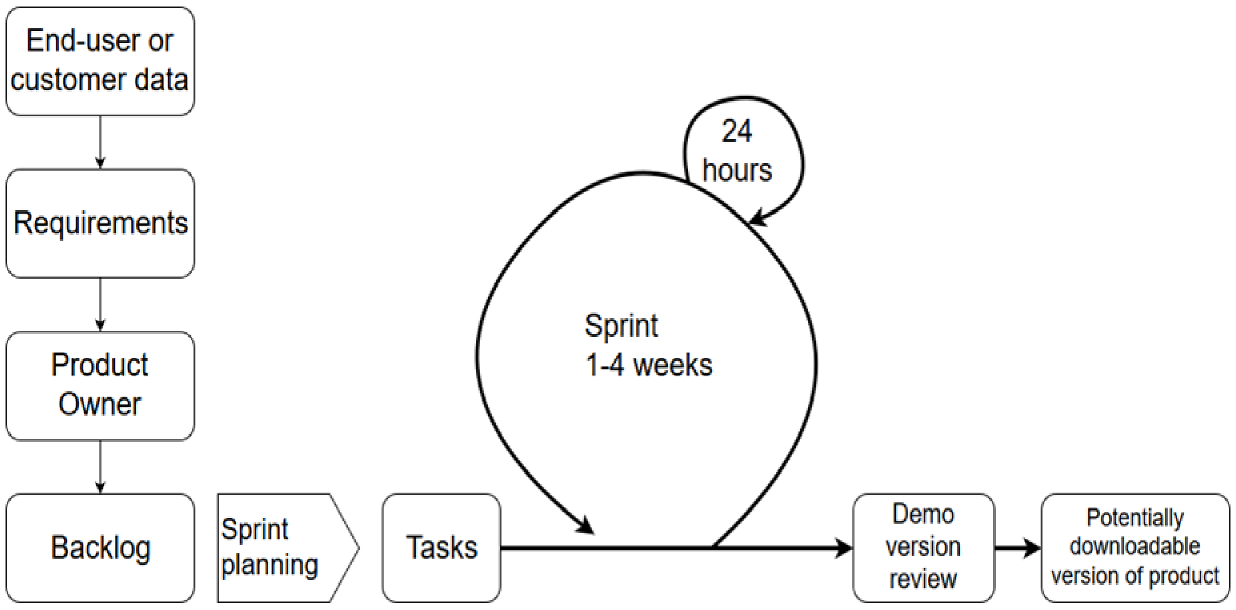


Fig. 2.2. Workflow scheme for scrum methodology.

Scrum sprint

Another Scrum's difference from Kanban is that the task priorities are defined not by the team but by the product owner, and all subsequent priority reviews during the project are performed by the product owner, not by the team. It's also worth paying attention to the sprint's composition - unlike the Kanban workflow, where all developers start from the beginning to work out the task and at completion take a new one, in Scrum each sprint begins with the planning, which conditionally could be named "sprint planning" stage. This stage usually takes the first few hours of sprint and team checks the list of development tasks because both the list and the priorities of tasks can be changed in time. Then the developers choose as many tasks with highest priority as the team could complete during sprint and start working. The bulk of the sprint can be named "realization", at this time the developers complete the tasks chosen at the planning stage. At last performed work is demonstrated to the product owner and integrated into the product – this stage can be named "release version" or simply "release". The sprint ends with a so-called "retrospective" - the team discusses the sprint, identifies which problems it has faced, whether they have been solved, how to improve the team's performance for the next sprint and how to improve the quality of the product. It is impossible to add tasks to the sprint even urgent and very important, therefore all unfulfilled tasks are returned to the task list for consideration at the next sprint planning stage.

Conclusions

The dependencies and general features of life cycle models have been determined and outlined above in "General particular features of lifecycle models". Advantages and disadvantages of software lifecycle models have been studied and analyzed, as well as the factors that have driven further searches and improvements of project management approaches and consequentially have led to the transition from life cycle models to "looped" approaches (iterative, incremental and spiral models).

Thus, this article emphasizes not only the need to introduce the latest developments in the field of work process organization to the Ukrainian IT sector but also points out, perhaps more importantly, the needless rejection of past achievements. Nowadays the problem of project management usually lies in the choice between Scrum and Kanban, not considering even other methodologies of Agile family, needless to say, the life cycle models. Those models, albeit outdated, still have their advantages in particular cases and moreover can sometimes surpass benefits of Agile methodologies.

ЛІТЕРАТУРА

1. Максим Пименов. Как создаются программы по методологии Agile. *Нетология-групп*, 2011-2018. URL: https://netology.ru/blog/loveagile. (Дата звернення: 11.03.2018)
2. Igor Mats. Модели жизненного цикла программного обеспечения. *Habr*, 2006 – 2018. URL: https://habr.com/post/111674. (Дата звернення: 10.03.2018)
3. Лилия Хаф. Методологии разработки программного обеспечения Часть 1. *КомпьютерПресс*, 1999 – 2018. URL: https://compress.ru/article.aspx?id=11321#05. (Дата звернення: 17.03.2018).
4. Thomas Alva. Ещё раз про семь основных методологий разработки. *Habr*, 2006 – 2018. URL: https://habr.com/company/edison/blog/269789. (Дата звернення: 15.04.2018)
5. Методологии разработки программного обеспечения. *Habr*, 2006-2018. URL: https://habr.com/sandbox/43802. (Дата звернення: 24.03.2018)
6. Agile Manifesto. *Manifesto for Agile Software Development*, 2001. URL: http://agilemanifesto.org. (Дата звернення: 07.04.2018)
7. Гибкая методология разработки (Agile). *Mahamba*. URL: http://mahamba.com/ru/gibkaya-metodologiya-razrabotki-agile. (Дата звернення: 08.04.2018)
8. Максим Пименов. Разбираемся в Scrum и Kanban. Нетология-групп, 2011-2018. URL: https://netology.ru/blog/scrum-kanban. (Дата звернення: 11.03.2018)
9. Александр Пушкарев. Методология Kanban: введение. Habr, 2006-2018. URL: https://habr.com/post/230725. (Дата звернення: 22.04.2018)
10. Mike McLaughlin. What Is Agile Methodology? VersionOne, Inc., 2018. URL: https://www.versionone.com/agile-101/agile-methodologies. (Дата звернення: 28.04.2018)
11. Маттиас Маршалл. КАНБАН / СКРАМ / АДЖАЙЛ — что лучше для вашего проекта? Brain Rain, 2017. URL: https://brainrain.com.ua/%d1%81%d0%ba%d1%80%d0%b0%d0%bc-2. (Дата звернення: 29.04.2018)
12. Agile Methodology: The Complete Guide to Understanding Agile Testing. QASymphony, Inc., 2018. URL: https://www.qasymphony.com/blog/agile-methodology-guide-agile-testing. (Дата звернення: 05.05.2018)
13. David Lowe. Theme, Epic, Story, Task. Scrum & Kanban Ltd, 2013-2018. URL: https://scrumandkanban.co.uk/theme-epic-story-task. (Дата звернення: 13.05.2018).

REFERENCES

1. M. Pimenov, “How programs are created using the Agile methodology”, December 2016. [Online]. Available: Netology-group, https://netology.ru/blog/loveagile. [Accessed March 11, 2018].
2. I. Mats, “Software life cycle models”, January 2011. [Online]. Available: Habr, https://habr.com/post/111674. [Accessed March 10, 2018].
3. L. Khaf, “Software Development Methodologies Part 1”, 2004. [E-book]. Available: ComputerPress, https://compress.ru/article.aspx?id=11321 [Accessed March 17, 2018].
4. T. Alva “Once again about the seven main development methodologies”, November 2015. [Online]. Available: Habr, https://habr.com/company/edison/blog/269789. [Accessed April 15, 2018]
5. “Software development methodologies”, April 2012. [Online]. Available: Habr, https://habr.com/sandbox/43802. [Accessed March 24, 2013].

6. “Agile Manifesto”, 2001. [Online]. Available: Manifesto for Agile Software Development, http://agilemanifesto.org. [Accessed April 15, 2018].
7. “Flexible development methodology (Agile)”. [Online]. Available: Mahamba, http://mahamba.com/ru/gibkaya-metodologiya-razrabotki-agile. [Accessed April 8, 2018].
8. M. Pimenov, “Understanding Scrum and Kanban”, December 2016. [Online]. Available: Netology-group, https://netology.ru/blog/scrum-kanban. [Accessed March 11, 2018].
9. A. Pushkarev, “Kanban Methodology”, July 2014. [Online]. Available: Habr, https://habr.com/post/230725. [Accessed April 22, 2018].
10. M. McLaughlin, “What Is Agile Methodology?”. [Online]. Available: CollabNet Inc., https://www.versionone.com/agile-101/agile-methodologies. [Accessed April 28, 2018].
11. M. Marschall, “Kanban vs Scrum vs Agile”, July 2015. [Online]. Available: Agile Web Development & Operations, https://agileweboperations.com/2015/07/27/scrum-vs-kanban/. [Accessed April 29, 2018].
12. “Agile Methodology: The Complete Guide to Understanding Agile Testing”. [Online]. Available: QASymphony Inc., https://www.qasymphony.com/blog/agile-methodology-guide-agile-testing. [Accessed May 05, 2018].
13. D. Lowe, “Theme, Epic, Story, Task”, January 2014. [Online]. Available: Scrum & Kanban Ltd, https://scrumandkanban.co.uk/theme-epic-story-task/. [Accessed May 13, 2018].

Головко Дмитро Сергійович - *магістрант, Харківський національний університет імені В.Н. Каразіна, майдан Свободи, 4, Харків-22, Україна, 61022; e-mail: ds.holovko@gmail.com; ORCID: 0000-0001-6134-3777.*

Holovko Dmytro S. *bachelor of computer science, V. N. Karasin Kharkiv National University, Svobody Sq 4, 61022, Kharkiv, Ukraine, ds.holovko@gmail.com; ORCID: 0000-0001-6134-3777.*

Головко Дмитрий Сергеевич- *магистрант, Харьковский национальный университет имени В. Н. Каразина, площадь Свободы, 4, Харьков-22, Украина, 61022; e-mail: ds.holovko@gmail.com; ORCID: 0000-0001-6134-3777.*

Васильєва Лариса Валентинівна – *кандидат біологічних наук, доцент кафедри електроніки і управляючих систем, Харківський національний університет імені В. Н. Каразіна, майдан Свободи, 4, Харків-22, Україна, 61022; e-mail: ds.holovko@gmail.com; ORCID: 0000-0001-7926-3062.*

Vasylieva Larysa V. - *PhD of biology, Associate Professor at the Department of Electronics and Control Systems, V. N. Karasin Kharkiv National University, Svobody Sq 4, 61022, Kharkiv, Ukraine, ds.holovko@gmail.com; ORCID: 0000-0001-7926-3062.*

Васильева Лариса Валентиновна – *кандидат биологических наук, доцент кафедры электроники и управляющих систем, Харьковский национальный университет имени В.Н. Каразина, площадь Свободы, 4, Харьков-22, Украина, 61022; e-mail: ds.holovko@gmail.com; ORCID: 0000-0001-7926-3062.*

Надійшла у першій редакції 27.08.2019, в останній - 03.09.2019.