ATLANTIS PRESS

Advances in Economics, Business and Management Research, volume 70

International Conference on Economy, Management and Entrepreneurship（ICOEME 2018）

Software Code Influence on Social Processes Within the Developers' Team: Creativity Aspects of Software Development

Dmitry Makarov
Bauman Moscow State Technical University
5/1, 2nd Baumanskaya Str.
Moscow, Russia 105005
E-mail: d.makarov@corp.mail.ru

Evgeniy Yusipov
Bauman Moscow State Technical University
5/1, 2nd Baumanskaya Str.
Moscow, Russia 105005
E-mail: yusipovea@bmstu.ru

Anna Sorotnik
Bauman Moscow State Technical University
5/1, 2nd Baumanskaya Str.
Moscow, Russia 105005
E-mail: anny.sorotnik@gmail.com

***Abstract*—In most cases, a large team of employees performs a modern software development. This team is a social group, during the work of which, there arise conflicts resulting from big differences between the working group members. The article considers the issues of software development paradigms influence on social processes within a developers' team as a mechanism for reducing the conflicts frequency between developers and a method of non-material employees' motivation.**

Keywords—development paradigms; conflicts in the team; staff motivation; social interactions; SOLID principles; software development

I. INTRODUCTION

During the software developers' teamwork, social processes within the working group play an important role. This team performing a large project often includes people of different sexes, age groups and it happens to embrace a significant number of foreign participants. The people in this social group will inevitably have different ambitions, educational level, and accumulated experience; have different views on the assigned tasks optimal solution. We should also note that for technically complex projects, it is necessary to attract different specializations' employees, and in these mixed groups, people sometimes spend much of the time on interactions between each other. Conflicts resulting from developers' team diversity can also result from the impossibility to choose a variant of development technical task execution satisfying all [1-3].

Such conflicts can lead to a work schedule disruption, a decrease in task quality, an increase in project cost, or even to the project closure; because colleagues may lose time allotted for the idea implementation, as they could not come to some consensus [4].

The international consulting company "The Standish Group International" has been analyzing the software development industry since 1994. In 2015 50,000 projects of different scales were studied all around the world. In the report "The CHAOS Reports" the company shows that only 29% of the projects have been successful; 52% of projects have come to an end late, expenses exceeded the planned budget, not all functions of the product were realized; 19% of projects failed and were closed. Also, the report indicates the factors that influenced success, such as Skilled staff, Agile Process, Project Management Expertise, etc., but social processes within the development team were not considered at all [5].

Also, it is necessary to note the specifics of software development in comparison with any other production activity. What programmers produce is immaterial - this is the result of collective thought, written in the programming language. The manager is not enough to use only project management techniques, such as Agile or PMBoK. It is necessary to pay special attention to the process of writing code and the interaction between programmers.

Thus, it is necessary to evaluate regularly the social indicators in the developers' team and integrate the techniques that facilitated an effective decision-making by all team members during the development process. In addition, at the project initiation stage, we should pay attention to team organization forms and employees motivation [6]. This need is justified by the collective work nature, which makes it difficult to take into account the individual programmer contribution and complicates the work planning.

II. Creativity Aspects of Software Development

The creativity is something that brings together programming with science and art. But the creative process of the programmer's work is fundamentally different from creation in science and art. Modern programs are created by a large number of developers, while scientists and artists work usually alone. For the collective creativity of software developers, an analogy with staging a theatrical performance or filming a movie is appropriate. In these processes a large number of specialists from different industries are involved, there is an inherent creative component, the correct recruitment of the team is very important, the number of failed projects is hardly less than in software development projects. Walker Royce, the authority in the field of project management, writes in his works: "Software project managers are more likely to succeed if they use techniques that are more like managing a movie production than an engineering production". [7]

Nevertheless, programming is not science or art. Programming does not have a clear system of knowledge about the laws of creating programs, it is also not a creative reflection of reality in artistic images. Programming is a craft in which the quality of the result depends on the skill of the developer. The number of development team members matters only to a certain extent. Frederick P. Brooks, an outstanding researcher in the theory of computer systems, argued that an attempt to solve the problem of delay during the program project deadline by adding additional staff to the team would only slow down the project[8]. To create a high-quality software product, it is not enough to learn how to write software code and follow clear instructions. The quality depends on the skill of each member of the team and their coordinated work.

III. Efficiency Indicators and Achievement Methods

As a measure of the software development efficiency, we take into account not only the development cost and each project stage performing time, but also the software qualitative characteristics: scalability, modifiability, and the ease to understand the program by a person. Among unsuccessful project signs, it is common to emphasize the following [9]:

- Non transformability: the system developed is difficult to change, because any attempt to change something causes the «snowball» effect affecting other system components.
- Instability: to make changes in any part of the project code means to disrupt the system operation in other blocks that have no relation with the fragment that is being changed.
- Stillness: the complexity or impossibility to decompose the program code into components that could be used to develop other projects.
- Viscosity: making changes without violating the project principles requires a lot of time and effort.
- Unjustified complexity: there are components in the project that do not bring expected benefits.
- Uncertainty: the difficulty in reading and understanding the project code.

Criteria for the development efficiency oppose to each other, because when trying to write the most understandable and easily scalable program code, the development time inevitably increases, and vice versa. In this regard, you can divide all programmers into two large categories [10]. The first category tends to perfectionism during development process. Regardless of the technical task, this group tries to write the program code, taking into account the possible product modification in the future, and tries to make the code as understandable as possible for the software further support.

The second programmers' category considers the work done on time as the most important task and does not bring to the forefront the code quality, they believe it does not play a key role in the work process. The very presence of two programmers' categories in the developers' team inevitably leads to the inconsistency. Evidently, because of this the project performing speed and final product quality are suffering, and development cost is increasing.

To deal with conflicts, software development paradigms are created, i.e. functional programming, imperative programming, structured programming, object-oriented programming, etc. Each paradigm represents certain sets of requirements for how the software development process should proceed [11].

If we consider functional programming paradigm, the developer is invited to present the program in the pure functions form and functions of higher orders, using, depending on the programming language, those or other collections for data storage. In imperative paradigm, the program is presented as a set of grouped data (structures), functions and procedures for their processing. However, in object-oriented programming paradigm, which now is one of the most common in the industrial environment, there is no strict descriptions of how the development process should be conducted [12]. It brings to the forefront the relations between objects in the system. Thus, the programmer task when using this paradigm becomes the task decomposition into domains of entities and these entities description. The problem is that this approach gives a great deal of variation regarding the architecture application choice.

Thus, it is not possible at the development initial stages to determine the system division into modules and the relation between them. Such uncertainty can lead to the fact that the final implementation may not be modifiable and not scalable, since the number of possible options for establishing connections between modules within the program is large [13]. It is such a variability in how you can implement the system being developed, that leads to conflicts in the programmers' team.

Some of programmers want to spend more time developing the application architecture to make it as elegant as possible in technical terms. Others are in a hurry to

implement the initial project functionality and rather start writing code key blocks. To simplify the application architecture development in the object-oriented programming paradigm, the SOLID principles were proposed, adhering to which the programmer will receive a modifiable and scalable application architecture.

IV. SOLID PRINCIPLES AND THEIR IMPLEMENTATION TO THE SOFTWARE DEVELOPMENT PROCESS

SOLID in programming is a mnemonic acronym introduced by Michael Feathers for the five object-oriented basic principles and design proposed by Robert C. Martin. To meet the SOLID principles is to be an ideal state for a software product. The software should strive for this state [14]. "Table I" shows the decoding of the term, where each letter of the abbreviation stands for a specific principle.

TABLE I. ALGORITHM OF THE TEAM WORK USING THE SOLID PRINCIPLES

Initial	Value	Name and description of the principle
S	SRP	«The Single Responsibility Principle». There is only one reason, which leads to the appearance of a class.
O	OCP	«The Open Closed Principle». Program entities must be open for expansion, but they are closed for modification.
L	LSP	The «Liskov Substitution Principle». Objects in the program must be replaced by instances of their subtypes without changing the correctness of the program.
I	ISP	«The Interface Segregation Principle». Many interfaces specifically designed for customers are better than one general-purpose interface.
D	DIP	«The Dependency Inversion Principle». Dependence on Abstractions. There is no dependence on anything concrete.

There are some difficulties associated with application of these principles in practice. If the code does not match to a SOLID principle, this may be a consequence of both developer insufficient professional training, and a lack of time to study the architecture. If you do not yet have time to refactor the code, architectural errors can stay in the application for a long time. The software product success depends on how the developers choose the entities within the framework of the task and what relationships will be established between them [15]. It may turn out that the chosen architectural design is easy to implement in the short term, but is impossible to implement in the long term, because in the future, it will be difficult to modify the class hierarchy and impossible to scale the program from the point of view of calculations. Therefore, we must fix the well-established practices that would lead to successful results in the development process. Although there exist some formulations to describe in general architecture concept development for the object-oriented program [16], this specific concept application falls on the shoulders of an architect-developer. Thus, the concepts formulations themselves already allow all the developers team to choose which way to follow while implementing some functionality. The recognition of the fact that the in the project it is prohibited to violate the above principles makes programmers take such technical decisions that would not violate these statements. This in turn will reduce the number of conflicts that could arise in the team. To implement the SOLID principles in the software development process, it is necessary that the team should understand them and share them. One possible option for implementing this technology is the two-step program code fragments addition to the working project. Thus, at the first stage the programmer performs his task in accordance with the SOLID principles, and then, before finalizing the code in the project, sends it to his colleagues for approval. In turn, his colleagues check the code whether it complies with the task and meets the SOLID basic principles. Further, the code either is sent for revision or is included in the main part of the project. Such a team interaction simple scheme "Fig. 1" provides not only good project quality, but also a team training through feedback within the team. As a result, a well-written code contributes to the favorable working atmosphere because if there arises a need for an urgent modification, there will be no cross-claims between project participants, any code fragment will be easy to modify and understandable to each programmer, such a code is quick and easy to work. Therefore, any programmer can feel their importance in the team and realize their involvement in a great quality project, though carrying out small and scattered tasks within the project. In addition, it is necessary to take into account the fact that the programmer work is associated with high-level abstractions, which requires great intellectual and emotional costs. The application of an architecture, built using the SOLID principles, allows you to reduce work tension indicators, which in turn increases the efficiency, and improves a psychological state of a person.

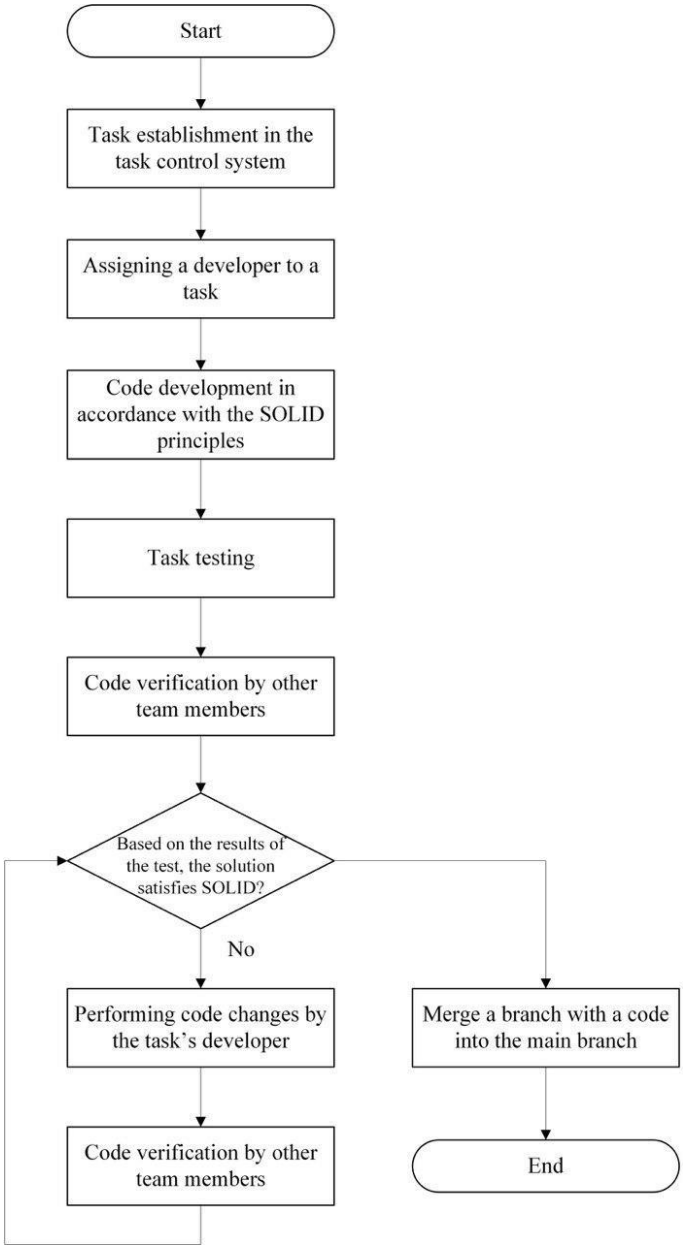

Fig. 1. Algorithm of the team work using the SOLID principles.

V. CONCLUSION

Not only technical aspects but also, largely, social aspect of programming principles affects the quality, cost and terms of the final software product implementation. A set of proven principles and architectural solutions is extremely necessary to apply in the program development, as this will affect not only the software development technical process, but also the team members' social interaction. Using proven solutions programmers will create a code that will not violate the SOLID principles, will be scalable and structurally understandable for software product further support. This will also speed up the project and reduce the development cost. Thus, both categories of programmers in the developers' team get an opportunity to pursue their interests, which will greatly improve the development process organization and will create favorable conditions for the team to work on future projects.

REFERENCES

[1] Chernikov B.V., Antonchikov S.N. Modeling of adaptive organization, 2017 Tenth International Conference Management of Large-Scale System Development (MLSD), Moscow, 2017, pp. 1-4.DOI:10.1109/MLSD.2017.8109605

[2] Karasev V.O., Sukhanov V.A. Product lifecycle management using multi-agent systems models, Procedia Computer Science, 2017, Vol. 103, pp. 142-147.DOI:10.1016/j.procs.2017.01.034

[3] Yadgarova Y.V., Taratukhin V.V., Skachko E.N. Multi-agent Product Life Cycle Environment. Interoperability Issues, Lecture Notes in

Business Information Processing, 2015, Vol.213, pp. 101-112.DOI:10.1007/978-3-662-47157-9_10

[4] Morozov A. A., Vaish A., Polupanov A.F. Development of Concurrent Object-Oriented Logic Programming Platform for the Intelligent Monitoring of Anomalous Human Activities, Communications in Computer and Information Science, 2015, Vol. 511, pp. 82-97.DOI: 10.1007/978-3-319-26129-4_6

[5] Lynch J. The Standish Group Chaos Report, 2015

[6] TishinaE.A., RezantsevaE.Ya., Reut D.V.The concept of digital transformation of the society, 2017 Tenth International Conference Management of Large-Scale System Development (MLSD), Moscow, 2017, pp. 1-5.doi: 10.1109/MLSD.2017.8109697

[7] Walker Royce. Successful Software Management Style: Steering and Balance. IEEE Software, 2005, Vol. 22, No.5, pp. 40-47

[8] Brooks F.P. The Mythical Man-Month: Essays on Software Engineering, 1995. — P. 336

[9] Martin R.S. Rapid program development: principles, examples, practice, Moscow, Williams Publishing House Publ., 2004.

[10] Rainwater, H. Herding Cats A Primer for Programmers Who Lead Programmers,Apress Publ., 2002.

[11] Erich Gamma, Richard Helm, Ralph Johnson, John Vlylesses Methods of object-oriented design. Design Patterns, Saint Petersburg, Peter Publ., 2001, 368 p.

[12] Liskov, Barbara, Wing, Jeannette Behavioral Subtyping Using Invariants and Constraints - CMU-CS-99-156, July 1999.

[13] Martin R. C. The Interface Segregation Principle, C ++ Report Publ., June 1996.

[14] Martin R. C., Clean Code: A Handbook of Agile Software Craftsmanship, Pearson Education Publ., 2008.

[15] Martin R. C. The Open-Closed Principle, C ++ Report Publ., January 1996.

[16] Martin R. C. The Dependency Inversion Principle, C ++ Report Publ., May 1996.