BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **20**, No 4
Sofia • 2020 Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.2478/cait-2020-0046

On the Usability of Object-Oriented Design Patterns for a Better Software Quality

Boyan Bontchev[1], *Emanuela Milanova*[2]

[1]*Dep. of Software Engineering, Sofia University "St Kliment Ohridski", 1504 Sofia, Bulgaria*
[2]*Nemetschek Bulgaria, 1202 Sofia, Bulgaria*
E-mails: bbontchev@fmi.uni-sofia.bg EMilanova@nemetschek.bg

***Abstract*:** *Software design patterns incarnate expert knowledge distilled from the practical experience in object-oriented design, in a compact and reusable form. The article presents a quantitative study of the usability of the object-oriented software design patterns* (*known as Gang of Four patterns*) *applied for improving the testability, maintainability, extendibility, readability, reliability, and performance efficiency of software applications. We received 82 usable responses from software professionals in Bulgaria, with 65 of them addressing both the usability and recognition of each one of the Gang of Four patterns, together with their impact on important software quality characteristics. As well, we studied the approach of each software developer in choosing a particular design pattern to use in order to solve a problem. We found statistically significant differences between the most recognized and most useful patterns and between the most unrecognized and most useless patterns, split into creational, structural, and behavioral groups.*

***Keywords*:** *Design patterns, usability, software quality, survey*.

1. Introduction

Since their invention in the middle of the nineties of the last century, software design patterns continue having a crucial impact on the discipline of software design. These patterns "preserve design information by capturing the intent behind a design" [1] and represent sophisticated and reusable solutions to recurring problems in the design of software applications. A design pattern is an abstraction of the particular form with a common purpose that maintains in repetitive specific and non-random contexts and can be applied to a particular solution [2]. Therefore, software design patterns incorporate schemas of expert knowledge elicited from the practical experience of software engineers while encountering similar types of problems regarding design and development of software platforms and applications [3].

Object-Oriented (OO) design patterns facilitate the reuse of successful designs and architectures of object-oriented software systems [4]. The most popular OO designed patterns are proposed by G a m m a et al. [1] and are widely known as *"*Gang of Four*"* (abbreviated to GoF) patterns. The GoF patterns are divided into three groups: (1) Creational (denoted below as C) – related to the creation of objects

in the most appropriate way to the given context, (2) Structural (S) – facilitate software design by identifying an easy way to realize relationships between objects, and (3) Behavioral (B) – provide common communication models between objects and the implementation of these models. The patterns provide coding of expertise and best practices in OO design and are independent of a specific language [5]. Therefore, software designers and developers apply them for decades as proven solutions offering many advantages regarding the most important characteristics of software quality, such as testability, maintainability, extendibility, readability, reliability, and performance efficiency [6].

Despite all the advantages listed over, there are studies showing that using design patterns can degrade the design quality by leading to the so-called *pattern coupling* [7] or by aggravating the overall maintainability due to an uncontrolled use of design patterns [8]. Another research reports a negative impact of applying design patterns on the complexity of the original design [9], or on reusability and readability [10]. On the other hand, H e g e d ű s et al. [11] have performed an extensive study dedicated to maintainability and found that the use of design patterns can improve any one of the ISO/IEC 9126 quality characteristic. Z h a n g and B u d g e n [12] have studied the usefulness of GoF design patterns reported by experienced software specialists and report some patterns to be highly used while others not. A l g h a m d i and Q u r e s h i [13] found the effect of design patterns on the software maintainability depends on the pattern size and the prior expertise of the software designers.

The goal of the present research is to study the usability of the object-oriented software design patterns (GoF patterns) applied for improving the software quality characteristics as perceived by professionals involved in the software industry of Bulgaria. This East European country has well-known traditions in information technology development and software excellence and, in last decades, has a software industry growth per year greater than 10% [14]. Due to its rapid advancing from basic outsourcing to modern engineering services and business consulting, software development is becoming more and more demanding for quality features of the products being created. Therefore, we decided to design a quantitative study about the usability of GoF design patterns for a better software quality and to conduct it in Bulgarian software companies designing and developing business applications. We found which are the most recognized and most useful design patterns and, as well, which are the most unrecognized and most useless ones. Another specific feature of our study is its orientation not only to highly experienced pattern users like in [12] but to designers and developers with less practice in patterns usage. However, the main difference with other studies is that we sought the effect of design patterns not only on one software quality characteristic like maintainability [11, 13] but on more quality characteristics such as testability, maintainability, extendibility, readability, reliability, and performance efficiency. In our survey, we tried to find out how designers and developers do appreciate the importance of these software quality characteristics and, next, how they evaluate the contribution of each GoF pattern to each characteristic. We conducted an online survey asking 36 closed questions and received 82 usable responses from Bulgarian software professionals, with 65 of them

addressing the impact of OO design patterns on software quality and the features of a certain pattern regarded by designers as most important for solving a problem. Thus, the answers allowed us to draw up valuable conclusions about the usability of object-oriented design patterns leading to a better software quality.

2. Research background

2.1. The object-oriented design patterns and their role in the software development

The design of object-oriented software is difficult, and the development of object-oriented software that can be reused is even more difficult. The design must be specific to the problem, but it should also be generic enough to be able to handle future problems and requirements. In addition, the design must not require redesign or at least the redesign must be minimized when the requirements change [16].

Software design patterns realize the concept of architectural patterns coined by Christopher Alexander [17] about describing "a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice". Similarly, software design patterns represent forms that encode and externally shape schemas of the expert knowledge derived from the practice of software design [3]. Using proven techniques such as OO design patterns makes these successful designs more accessible to developers of new systems. Such design patterns help in selecting alternatives that make the system reusable and avoid alternatives that can compromise this possibility. OO design patterns can even improve the documentation and maintenance of existing software systems by providing a clear specification of interactions between classes and objects [1]. Simply said, templates help the object-oriented designers to get the "right" design faster and solve the day-to-day problems faced by software developers such as it is follows.

- Finding the right objects – the hardest job in the design of object-oriented software consists in decomposition of systems into classes and objects, which is hampered by design issues such as abstraction, data hiding, capsulation, and polymorphism [15] determining many contradictory factors like object granularity, coupling, cohesion, dependency, understandability, and reuse [18].
- Determining the granularity of the object representation of a software system at all granularity levels (package, class, attribute, and method) – the granularity has a crucial impact on other quality factors of the software [19]. For example, the Façade pattern describes how to represent entire subsystems as objects, and the Flyweight pattern allows to maintain a huge number of fine-grained objects [1].
- Creating an interface for an object – the interface of an object characterizes the full set of queries that can be sent to the object. Design patterns help to define interfaces by identifying their key elements and the types of data that are sent through the interface. Design templates also specify the relationships between the interfaces. In particular, they often require some classes to have similar interfaces or to restrict the interfaces of some classes. For example, both Decorator and Proxy templates require object interfaces to be identical to the decorated and proxy objects [5].

- Create a design that can be easily changed – in order to design a system so that it is resistant to further changes, one needs to think about how the system may need to change in the future. These changes may include redefining classes and re-implementation, client change, and re-testing [20]. Common causes of redesign can be creating an object by explicitly defining the class (which engages with a particular implementation instead of a specific interface), hardware and software platform dependence, dependence on the object signature or its implementation, impossibility to easily change the classes, and others [11].

2.2. Existing research on the impact of design patterns on software quality

Many studies present design patterns as promising solutions to improve the quality of object-oriented software systems during development. They are said to improve the quality of the systems and it is also said that all well-structured object-oriented architectures contain patterns [1]. Some studies, however, show that using design patterns does not always lead to a good quality design. In particular, the intricate application of these design patterns has a negative impact on the quality that these patterns are said to improve [7]. In addition, design patterns typically increase the complexity of the original design to facilitate future enhancements [9].

W y d a e g h e et al. [21] have presented a study on the specific use of six design patterns for building an OMT/UML editor. They discuss the impact of these models on quality attributes such as reuse, modularity, flexibility, and readability. They conclude that although design patterns have many advantages, not all patterns have the same effect on quality attributes and that the developer needs a lot of professional experience with them in order to easily apply them to a particular application in their full potential. However, this study is limited to the authors' own experience and thus their subjective assessment of the impact of these models on quality can hardly be summed up and applicable to every context of development. 10 years later, K h o m h and G u e h e n e u c e [10] conducted a study on the impact of design patterns on the quality attributes of a software. This empirical study was conducted by asking respondents for their impact assessments of all design patterns on certain quality attributes. They came to the conclusion that, contrary to popular belief, design patterns do not always improve reusability and readability, but they definitely improve costs. However, they acknowledge that this study is based only on a surveyed group of 20 experienced developers and therefore it cannot be assumed that its results are statistically significant, representative and definitive.

T a h v i l d a r i and K o n t o g i a n n i s [22] explore the 23 design patterns described in [1] and present a classification of layers of the basic relations between these patterns: use, reflection and conflict, and three secondary relations: similar, combined and obligatory. They organize design patterns on two levels of abstraction - primitive and sophisticated design patterns. They believe that their classification may help software engineers to better understand the complicated relationship between patterns. However, they are not exploring whether the use of these models really improves the quality of the design. On the other hand, M c N a t t and B i e m a n [7] explore the coupling between design patterns. They make a parallel between modularity and abstraction in software systems and, on the other hand,

modularity and abstraction in patterns. They also classify the groups of patterns by dividing them into two groups that are weakly linked and strongly linked. They conclude that when patterns are loosely coupled and very abstract, they contribute to easy maintenance and refactoring and the possibility for reuse.

Several studies on the use of design patterns are focused on one crucially important quality of the software design – maintainability. H e g e d ű s et al. [11] have analyzed over 300 revisions of JHotDraw, a Java work framework, whose design relies heavily on some well-known design patterns. In their study, they focus on two main research questions: (1) are there any traceable effects from the application of design patterns on software maintainability; (2) what is the relationship between the number of design patterns used and the maintainability. They conclude that any ISO/IEC 9126 quality characteristic, including maintainability, is evenly improved by the number of design patterns used. Another interesting result is that the density of the patterns line and the maintainability values have a very similar trend. At the same year, Z h a n g and B u d g e n [23] conducted a meta-study on the effectiveness of software design patterns. As a result, they found "some support for the usefulness of patterns in providing a framework for maintenance". In their next study [12], the same authors researched the usefulness of GoF design patterns reported by experienced pattern users considering software development and maintenance. They found the Observer, Composite, and Abstract Factory patterns to be highly used and, on the other hand, received differing views about Visitor, Singleton, and Façade. A l g h a m d i and Q u r e s h i [13] identified some specific issues influencing this effect like the pattern size and the prior expertise of the software specialists.

3. Research method

3.1. Research questions

The purpose of the research is to design and conduct a quantitative study for determining the usability of object-oriented design patterns among the software specialists in Bulgaria with focus on achieving a better software quality. Thus, the main research question can be formulated as follows: *How software professionals in Bulgaria apply object-oriented design patterns for improving software quality?* Its answer has to address the following issues:

- How many Bulgarian software professionals know what OO design patterns are and do they use them?
- Which design patterns are most famous and used; what are the differences in their ratings?
- Which design patterns are most unrecognized and useless; what are the differences in their ratings?
- Which software qualities are the most important for Bulgarian software developers?
- What is the contribution of OO design patterns to software quality characteristics most important for Bulgarian software professionals?

3.2. Survey design

For conducting the survey, we developed a questionnaire containing 36 questions divided into four different logical sections:

- Demographic data (9 questions) – this section aims to gain insight into the experience with software technologies and design patterns of the respondent, as well as gather demographics data such as work experience, education level, age and more.
- Usability of design patterns (5 questions) – this section aims to determine how well known are different object-oriented design patterns among people involved in software technologies and to derive which of the patterns are considered to be the most useful and most useless.
- Software quality characteristics (19 questions) – this section aims to determine what software qualities are most important to people who are involved in software technologies. In addition, it gains insight into the time each developer spends on implementing new features, improving quality or implementing new design patterns.
- Design patterns searching (3 questions) – this section aims to determine the approach of each software developer in choosing a specific design pattern for a problem he has to solve. In addition, it aims to understand whether the people will be open to trying new design patterns if they know it will improve the quality of their application.

We used dichotomous questions as well as closed questions to collect demographic data. For the other three sections of the survey, we used closed questions so we can easily process and analyze them. For most closed questions, we applied a five-point Likert scale. We used the following Likert scales:

- For measuring the usability of the various design patterns: 1 – "I'm not familiar with this pattern", 2 – "I'm familiar with this pattern, but I have not used it", 3 – "I have no opinion", 4 – "I'm familiar with this pattern and I've used it", 5 – "I use it very often";
- For measuring the quality characteristics of the code people value most: from 1 – "Very important" to 5 – "Not important at all".

3.3. Procedure

The online survey method was applied to collect the needed data. We created a web-based online questionnaire using Google Forms. The online survey was sent to Bulgarian software companies involved in design, development, and support of business applications. It was conducted within three weeks in the autumn of 2018. All the respondents participated in the survey voluntarily and anonymously. After accomplishing the survey, they were able to see the results up to the moment.

4. Analysis of results

4.1. Respondents' profile

The survey was completed by 82 respondents, whereupon 65 responded to have used OO software design patterns. We start by analyzing the results of the demographic

data provided by the respondents using frequency distributions, correlations, and cross-tabulations. For processing and analyzing the gathered data, we applied IBM SPSS and Microsoft Excel with XLSTAT. From all the 82 usable responses, 61 were submitted by men and 21 by women, which resulted in a gender balance of 74.39% to 25.61%. This gender ratio appears much closer to the one reported by Eurostat for the Bulgarian ICT companies in 2017, which is 73.52% to 26.48%. The majority of the respondents (46%) were 18-25 years old, while 35% and 16% reported being aged 26-35 and 36-45 years, respectively. Therefore, the results we obtained on the usability of the design patterns reflect the trends in software technologies over the last 10-15 years.

Fig. 1 shows the distribution of the respondents by profession. The most widespread professions among them are developer, intern student, and architect. From this distribution, combined with the previous one, we can deduce the following conclusion: In the last 10-15 years, the most occupied professions in the field of software technologies are developer (i.e., programmer) and architect (i.e., software designer) and the least occupied are maintenance and integrator. On the other hand, most of them reported using Object-Oriented Programming (OOP) languages such as C^{++}/C$^{\#}$, Java, Python, or PHP.

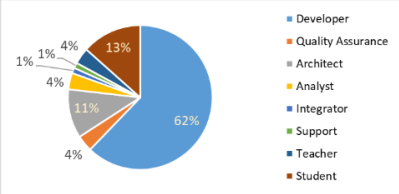

Fig. 1. Frequencies by professions

According to the years of experience with design patterns of the respondents, we could classify our respondents into three larger groups: (1) 50% of respondents reported initial experience with design patterns, i.e., less than 3 years; (2) 38% of them declared average experience with design patterns, i.e., between 3 and 10 years including; (3) 12% announced having high experience with design patterns, i.e., more than 10 years. The conclusion that we can deduct from this distribution is that design patterns have become more used and recognizable in Bulgaria in the last few years.

Table 1. Cross-tabulation between professional experience and design patterns experience

How many years of experience do you have with working with design pattern?		Which ansver best describes your primary role during software development?								Total
		Developer	Quality assurance	Architect	Analyst	Integrator	Support	Teacher	Student	
0-1	Count	11	2	1	1	0	1	0	11	27
	% within "Which answer best describes your primary role during software development?"	21.6%	66.7%	11.1%	33.3%	0.0%	100%	0.0%	100%	32.9%
1-3	Count	13	0	0	0	1	0	0	0	14
	% within "Which answer best describes your primary role during software development?"	25.5%	0.0%	0.0%	0.0%	100%	0.0%	0.0%	0.0%	17.1%
3-5	Count	13	1	1	0	0	0	0	0	15
	% within "Which answer best describes your primary role during software development?"	25.5%	33.3%	11.1%	0.0%	0.0%	0.0%	0.0%	0.0%	18.3%
5-10	Count	10	0	3	2	0	0	0	0	15
	% within "Which answer best describes your primary role during software development?"	19.6%	0.0%	33.3%	66.7%	0.0%	0.0%	0.0%	0.0%	18.3%
10-15	Count	4	0	3	0	0	0	3	0	10
	% within "Which answer best describes your primary role during software development?"	7.8%	0.0%	33.3%	0.0%	0.0%	0.0%	100%	0.0%	12.2%
15+	Count	0	0	1	0	0	0	0	0	1
	% within "Which answer best describes your primary role during software development?"	0.0%	0.0%	11.1%	0.0%	0.0%	0.0%	0.0%	0.0%	1.2%
Total	Count	51	3	9	3	1	1	3	11	82
	% within "Which answer best describes your primary role during software development?"	100%	100%	100%	100%	100%	100%	100%	100%	100%

Table 1 presents the relationship between the profession and the experience with design patterns through cross-tabulation. Most years of experience with design patterns are found in the architect and developer groups. Cross-tabulation has also been done between profession and OOP experience. The difference is that the experience of the respondents with OOP is a few years larger than the one with design patterns. These results lead to the natural conclusion that there is a strong relationship between experience with OOP and with design patterns. Indeed, the result of the correlation analysis reveals a strong, statistically significant correlation between the experience with OOP and with design patterns – the Pearson correlation coefficient appeared to be 0.856. This very high correlation shows that design patterns are a vital part of OOP. When people start using OOP, they eventually find a problem that can be solved by means of design patterns.

Another strong correlation regarding the demographic data was found between the age of the participants and their experience with design patterns. The results of the correlation analysis show that there is a correlation of 0.65374 statistically significant at p=0.000 for 2-tailed distribution. The respondents aged between 26 and 45 years reported experience with design patterns for more than 10 years.

Table 2. Cross-tabulation between profession and usage of design patterns

Have you ever used design patterns?	Which ansver best describes your primary role during software development?								Total
	Developer	Quality assurance	Architect	Analyst	Integrator	Support	Teacher	Student	
Yes	45	1	8	3	1	0	3	4	65
No	6	2	1	0	0	1	0	7	17
Total	51	3	9	3	1	1	3	11	82

Table 2 represents a cross-tabulation between profession and usage of design patterns. It reveals that the relative share of respondents who have never used design patterns is higher for intern students, testers (quality assurance), and support specialists. The analysis of the next sections of the survey continues with the 65 respondents (79.27% of all the respondents) who ever used design patterns.

4.2. Patterns usability

The second section of the survey aims to measure the usability, usefulness, and recognition of design patterns. As explained above, 65 of the respondents answered positively the question “Have you ever used design patterns?” and, thus, we transferred to this section. The question of measuring usability and recognition is as follows: “What is your experience with the following design patterns?” Responses are based on a five-point Likert scale as described in 3.2. For the analysis, we chose to assign the first two replies as a factor of unrecognition (i.e., the lack of use of the pattern is treated as unrecognition) and the last two as recognition. The middle answer (having no opinion about the pattern) is not counted. This is how we get the most recognized and unrecognized patterns, according to the respondents. Next, all the data were checked whether they had normal distribution or not. The Shapiro-Wilk, Lilliefors, and Jarque-Bera normality tests return p-value greater than the alpha level of 0.05, so we could not reject the null hypothesis and consider that the data are normally distributed.

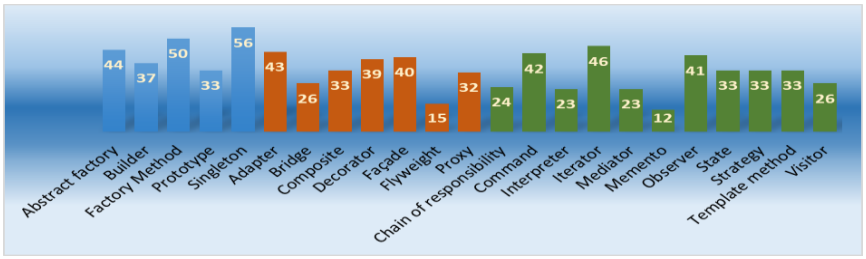

Fig. 2. Most recognized design patterns

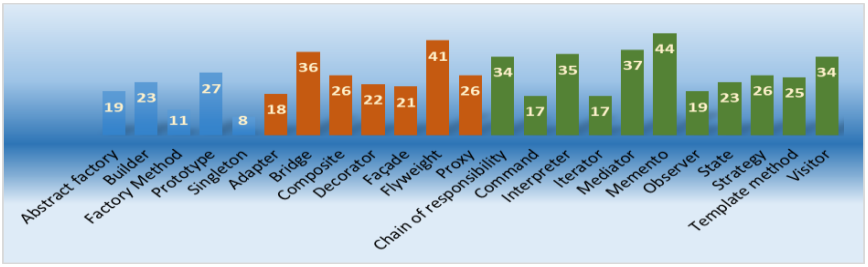

Fig. 3. Most unrecognized design patterns

Fig. 2 shows the distribution of the most recognized patterns (Mean M=34.0870, Standard Deviation SD=10.8121, and Standard Error SE=2.2545) ordered in three groups presented respectively in blue, brown, and green color – Creational (C), Structural (S), and Behavioral (B) patterns. The values for each of these are the number of respondents who have responded as "I know this pattern and I've used it" or "I use it very often" for that pattern. As we can see from Fig. 2, the five most recognized patterns are: Singleton (C) pointed by 56 (i.e., 86.15%) of all the respondents answered the pattern usability section; Factory method (C) – pointed by 76.92%, Iterator (B) – 70.77%, Abstract factory (C) – 67.69%, and Adapter (S) – 66.15%.

Fig. 3 represents the distribution of the most unrecognized design patterns (M=25.6087, SD=9.3164, SE=1.9426) ordered in three groups in the same way as done in Fig. 2. The values for each of them represent the number of respondents who selected "I'm not familiar with this pattern" or "I'm familiar with this pattern but I have not used it". Hence, the five most unrecognized design patterns are: Memento (B) pointed by 67.69% of all the respondents answered the pattern usability section, Flyweight (S) – pointed by 63.08%, Mediator (B) – 56.92%, Bridge (S) – 55.38%, and Interpreter (B) – 53.85%.

The other two questions in the second section of the survey are designed to measure the level of usefulness of design patterns. The two questions are as follows: (1) which design patterns do you think are most useful in your day-to-day work; (2) which design patterns do you think are mostly useless in your day-to-day work. From the answers to these two questions, we get the distribution of the most useful and useless design patterns according to the respondents. Fig. 4 presents the distribution of the most useful design patterns (M=16.6522, SD=9.8977, SE=2.0638). The values of each of them represent the number of respondents who have indicated the pattern as one of the most useful to the question Which design patterns do you think are most useful in your day-to-day work? In this way, we select the most useful

patterns at first five positions according to the respondents, as follows: Factory Method (C) – selected by 55.38% of all the respondents answered the pattern usability section, Singleton (C) – selected by 50.77%, Observer (B) – 40.00%, Builder (C) – 36.92% and, at fifth position, Iterator (B), Decorator (S) and Composition (S) – all pointed by 35.38% of the 65 respondents.

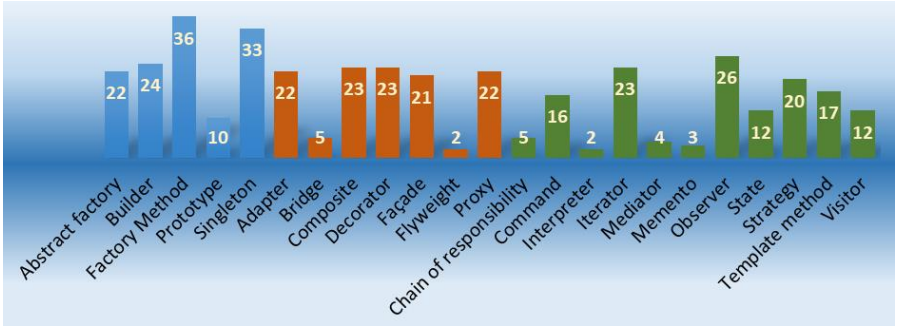

Fig. 4. Most useful design patterns

Fig. 5 shows the number of respondents who have indicated the pattern as one of the most useless to the question „Which design patterns do you think are mostly useless in your day-to-day work?" The basic distribution statistics here are M=6.8696, SD=4.5157, and SE=0.9416. Thus, we were able to select the five most useless patterns, according to the respondents: Flyweight (S) selected by 29.23% of all the respondents answered the pattern usability section, Memento (B) – selected by 23.08%, Visitor (B) – 20.00%, Singleton (C) – 18.46%, and Abstract Factory (C) – 13.85%.

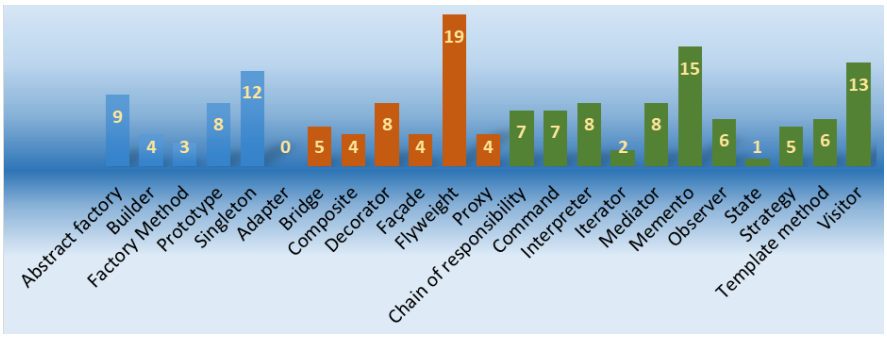

Fig. 5. Most useless design patterns

The distributions for the most useful patterns and the most recognized design patterns are quite similar. Bearing in mind the normal distribution of the collected answers, we applied Pearson's correlation as a statistical measure of the strength of a linear relationship between all the paired data. The Pearson's correlation between these two distributions appeared to be very high (0.89355, Table 3). We found very high negative correlations between the most recognized and most unrecognized patterns and, as well, between the most unrecognized and most useful patterns. At the same time, the correlation between distributions for the most unrecognized and most useless design patterns is 0.51303. This is much less than the previous correlation,

but we still see several patterns that are repeated in both distributions, such as Flyweight, Memento, and Visitor. People almost always recognize what they use in their practice and it is easy for them to determine whether it is useful or not. On the other hand, it is much harder to recognize uselessness – when you have not used something you can hardly determine whether it is useful or not. That could explain why the correlation between the most unrecognized and the most useless patterns is lower than the previous one.

Table 3. Correlations between reported usefulness and recognition of design patterns

Usefulness and recognition	Most recognized	Most unrecognized	Most useful	Most useless
Most recognized	1	–0.98699	0.89355	–0.49039
Most unrecognized		1	–0.90067	0.51303
Most useful			1	–0.43735
Most useless				1

Table 4. Comparison of most recognized, unrecognized, useful, and useless patterns' statistics

Comparison	Creational			Structural			Behavioral		
	M	SD	SE	M	SD	SE	M	SD	SE
Most recognized	44	9.3541	4.1833	32.5714	9.6412	3.6440	30.5455	10.0932	3.0432
Most unrecognized	17.6	7.9875	3.5721	27.1429	8.3751	3.1655	28.2727	9.0453	2.7273
Most useful	25	10.2470	4.5826	16.8571	9.1911	3.4739	12.7273	8.4272	2.5409
Most useless	7.2	3.7014	1.6553	6.2857	6.0749	2.2961	7.0909	4.1099	1.2392

Table 4 represents a comparison of the basic statistics (mean value, standard deviation, and standard error) about OO design patterns reported to be most recognized, unrecognized, useful, and useless. The statistics are split into three groups – for the creational, structural, and behavioral patterns. What we see from the results is that the most useful design patterns are from the creational group and the most useless are from the behavioral group. On the other hand, the results show that the most unrecognized and useless patterns are from the behavioral group, and the most recognized and useful ones are from the creational group.

In order to verify if the differences in the mean values found for the reported most recognized and most useful patterns are statistically significant, we performed paired T-test. Table 5 represents the differences of means of the most useful patterns and the most recognized patterns followed by the *p*-value of a paired lower-tailed T-test (all the differences are negative). Next, the table shows the same statistics regarding the most unrecognized and most useless patterns. The statistics for both comparisons are given for the three groups of patterns (creational, structural, and behavioral) and all appeared to be statistically significant. For these comparisons, Cohen's *d* can be used as an appropriate measure of the effect size, because the samples in both comparisons have similar standard deviations and are of the same size. The last row of the table represents Cohen's *d* for the most recognized versus most useful patterns followed by the most unrecognized versus most useless patterns – all split into three groups. The values reveal a high negative effect size for all of the comparisons.

Table 5. Differences between the most recognized and most useful patterns and between the most unrecognized and most useless patterns

Differences	Most recognized versus most useful patterns			Most unrecognized versus most useless patterns		
	Creational	Structural	Behavioral	Creational	Structural	Behavioral
$M_{difference}$	–19.0000	–15.7143	–17.8182	–10.4000	–20.8571	–21.1818
p ($T \leq t$, paired)	0.00055	0.00007	0.00000	0.03533	0.00003	0.00000
Cohen d	–1.93666	–1.66839	–1.91643	–1.67070	–2.85090	–3.01509

4.3. Quality characteristics and design patterns

The third section of the survey aims to determine what software qualities are most important to people involved in software technologies. The first question in this section is "Do you think using design patterns can improve the quality of the code?" and requires a "Yes" or "No" answer. From all the respondents, 90.8% think that design patterns improve quality and only 9.2% state the opposite opinion.

The second question is to find out which quality characteristics are most important. According to the respondents, the three most important quality characteristics are maintainability (selected by 76.92% of all the respondents), extensibility (72.31%), and reliability (46.15%); while readability, testability and performance efficiency were rated lower (Fig. 6).

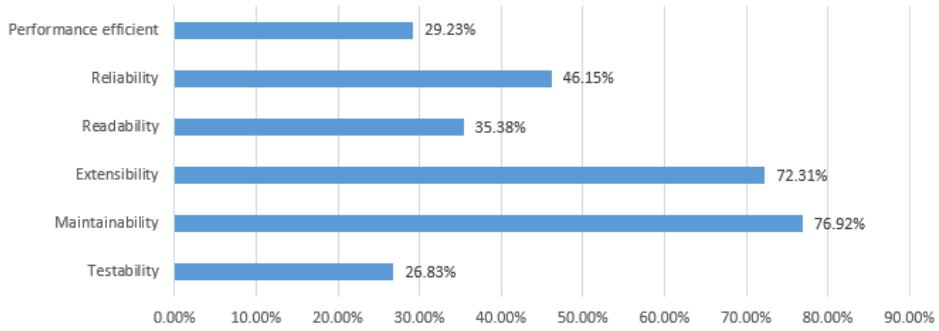

Fig. 6. Most important software quality characteristics according to the respondents (in percentages)

Fig. 7 shows the contributions of all the GoF patterns to the six software quality characteristics. The three patterns reported to have the greatest impact on software quality are Factory method, Iterator, and Façade. On the other hand, there were reported different patterns of contributions for each quality characteristic (after the pattern name follows the number of votes):

- Maintainability – Façade (20), Factory method (19), and Abstract factory (17);
- Extendibility – Abstract factory (22), Factory method (20), and Decorator (18);
- Reliability – Singleton (16), Iterator (10), and Interpreter (9);
- Readability – Façade (19), Decorator (17), and Builder, Factory method and Iterator (14);

- Testability – Abstract factory and Iterator (17), Adapter and Proxy (15), Builder and Strategy (14);
- Performance efficiency – Flyweight (13), Singleton (10), and Iterator (9).

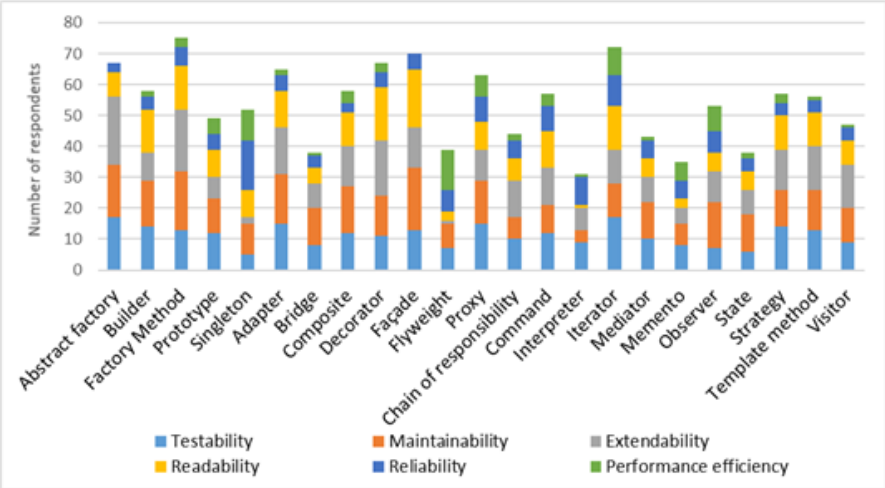

Fig. 7. Contributions of GoF patterns to the software quality characteristics according to the number of respondents

Obviously, the GoF patterns identified over are reported to contribute with different impact to software quality. The mean values of their impact on the software quality (Table 6) show they contribute more to maintainability, extendibility, testability, and readability, and less to reliability and performance efficiency. The GoF patterns contribute to the first three characteristics with the same impact because paired T-test calculated between the contributions of GoF patterns to the software quality characteristics show that the differences of means for the first three characteristics are not statistically significant. In Table 6, all the *p*-values given in bold show statistically significant differences. As well, the creational and structural patterns have a greater impact on quality than the behavioral ones; however, these differences appeared to be not statistically significant. Therefore, all the three pattern groups do contribute with the same impact on the software quality.

Table 6. Basic statistics and *p*-values for paired T-tests between the means of contributions of GoF patterns to the software quality characteristics

Paired T-tests	Basic statistics			$p(T \le t$, paired)				
	M	SD	SE	Maintain-ability	Extend-ability	Read-ability	Relia-bility	Performance efficiency
Testability	11.17391	3.41989	0.71310	0.16056	0.79330	**0.02950**	**0.00008**	**0.00000**
Maintainability	12.30435	3.87808	0.80864		0.13181	**0.00053**	**0.00002**	**0.00000**
Extendability	10.95652	5.13879	1.07151			0.10622	**0.00285**	**0.00020**
Readability	9.34783	4.52878	0.94432				**0.01016**	**0.00032**
Reliability	6.04348	2.86798	0.59801					**0.00108**
Performance efficiency	3.82609	3.48571	0.72682					

The next question aims to find out how often the respondents change an existing code to improve quality. The answers are again based on a five-point Likert scale. Fig. 8 shows the summary results for this question. The values of each answer

represent the number of respondents who indicated the answer. From the results, we see that the respondents change the code most often to simplify the code, create an interface or reduce coupling. These were also the most important quality characteristics according to them. Therefore, there is a logical relationship between the most important quality characteristics and the most common reasons for redesigning the code. The correlation between the reported importance of the quality characteristics of the code and the reasons for code redesign due to quality improvements was found to be 0.38857 regarding the size and complexity of methods and 0.43542 regarding the size and complexity of classes. The correlation reveals another logical relationship between the least important characteristics of the code and the rarest code changes, according to the respondents. The reasons that lead to the rarest code changes are the inheritance, code size reduction, and the use of design patterns.

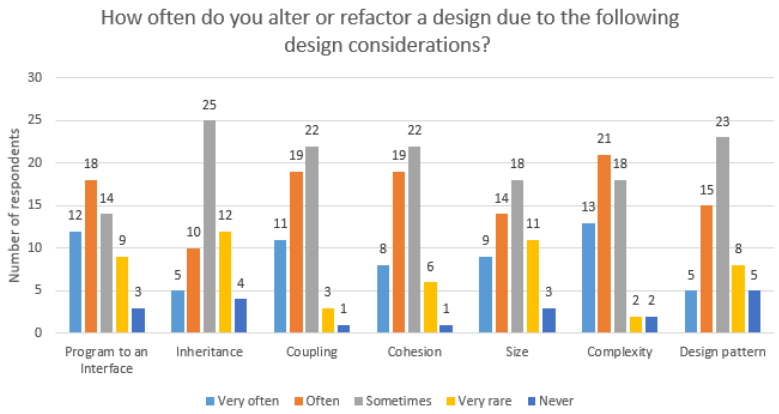

Fig. 8. Reasons for code redesign due to quality improvements according to a number of respondents

5. Discussion

5.1. On the results obtained from the survey

The results presented in the previous section could be discussed from various points of view. The first question when starting a data analysis is how is our sample different or the same as those in previous research studies. Unlike Khomh and Gueheneuce [10] and Zhang and Budgen [23], who researched the usefulness of GoF design patterns reported mostly by experienced pattern users, here we addressed a survey to software workers having different age, profession, and experience with design patterns. Almost a half of our respondents appeared to be young people being less than 25 years old and 40% of them reported having OO programming experience for less than 3 years. Half of the respondents had initial understanding of design patterns, i.e. less than 3 years. As well, we had circa two-thirds of them working as software developers and only 11% reported to be an architect (i.e., software designer). The age and the experience with both programming and design patterns appeared highly correlated. Thus, these demographic features of our sample correspond to the very rapid growth of the ICT industry in Bulgaria [14].

Young and inexperienced people with secondary education report they not used design patterns (20.73% of the sample). For the research of pattern usability, relationships of quality characteristics and design patterns, and the approaches for selection of design patterns we used the rest of the answers. Here, we included only closed questions because the closed type allowed us to conduct the survey easily and quickly and, next, to find the facts excluding an exhaustive qualitative analysis. We identified a very high correlation between the most recognized and useful GoF patterns. As a whole, the respondents use design patterns primarily for object creation and rarely look for elegant and trusted solutions for communication and relations between these objects. This explains why the most useful design patterns are from the creational group and the most unrecognizable are from the behavioral group. We found the five most recognized patterns (Fig. 2) are Singleton (C), Factory Method (C), Iterator (B), Abstract Factory (C), and Adapter (S); while the most useful patterns (Fig. 4) are Factory Method (C), Singleton (C), Observer (B), Builder (C), and Iterator (B), Decorator (S) and Composition (S). Therefore, Singleton (C), Factory Method (C), and Iterator (B) were found to be both most recognized and useful. Similarly, Z h a n g and B u d g e n [12] found the Observer (B), Composition (S) and Abstract Factory (C) patterns to be highly used – all these patterns we found being recognized and/or useful.

In our study, the five most unrecognized design patterns (Fig. 3) appear to be Memento (B), Flyweight (S), Mediator (B), Bridge (S), and Interpreter (B); while the five most useless patterns (Fig. 5) are Flyweight (S), Memento (B), Visitor (B), Singleton (C), and Abstract Factory (C). Therefore, Memento (B) and Flyweight (S) were found to be both most unrecognized and useless. Similarly, Z h a n g and B u d g e n [12] concluded that the patterns Flyweight (S), Singleton (C), Visitor (B), and Memento (B) are not considered useful – all these patterns we found being unrecognized and/or useless. On the other hand, they received differing views about Visitor (B), Singleton (C), and Façade (S). In our study, we do confirm the same for the Singleton (C) and Abstract Factory (C) patterns, which are in the distribution of the most recognized and most useful patterns but are also present in the distribution of the most useless patterns. Therefore, these results indicate a high degree of polarization of responses for these patterns. On the other hand, Visitor (B) was reported to be among the most unrecognized and useless patterns, while Façade (S) received moderate evaluation about its recognition and usefulness.

It is important to note that a pattern being liked by fewer developers does not mean it is less useful in general. In our survey we don't analyse the exact nature of the software applications our respondents are developing (e.g., banking, gaming, robotics, data-mining, mission critical, etc.). It might be that some of the "useless" design patterns are used more often in specific software application types from which we do not have enough representatives.

Probably our respondents have different productivity and therefore how often they use a given pattern might differ. One developer might use one pattern and produce as much code as other three developers who use another pattern. A more appropriate approach would be to use a "weighted" number of respondents but this will be too difficult to implement in reality.

Although the results from other questions showed that people surveyed mostly aim to ensure functional correctness and only then, they think about the quality of the created code, the respondents were strongly determined about software quality improvements resulted in applying design patterns. Regarding software quality, the most important things for them were reported to be the personal experience, code reviews, and design guidelines. Although the most unrecognized and useless set of patterns was from the behavioral group, and the most recognized and useful patterns were from the creational group, we found the three pattern groups do contribute with the same impact to the software quality. On the other hand, the GoF patterns were confirmed to contribute more to maintainability, extendibility, testability, and readability, and less to reliability and performance efficiency, in a statistically significant way. Respondents ranked code coupling, method (or class) complexity, cohesion, and interface building as more important than inheritance and lines per method (or class). These highly ranked quality characteristics of the code appeared among the most frequent reasons for code redesign due to quality improvements (Fig. 7). Although here design patterns are not pointed as first reason to alter or refactor the design, they improve strongly the quality characteristics of the code such as method (or class) complexity, code lines per method (or class), cohesion, coupling, inheritance and usage of interfaces [1].

5.2. On survey validity

When conducting a survey, there are three issues considered as potential sources of bias [12]: the design of the survey instrument; the way of administrating the survey; and the analysis of the data received. With regard of the design of the survey instrument, K i t c h e n h a m and P f l e e g e r [24] identified *content validity* to be the most important characteristic of the survey, due to representing the appropriateness of the instrument subjectively assessed by external reviewers having both knowledge and experience in the problem being studied. Before conducting the survey, we asked three external reviewers to do an informal assessment of our questionnaire and, next, reflected all of their remarks and comments raised about both the definition and representation of some questions. In order to assess the *criterion validity* of the questionnaire, we looked for similar surveys; however, we were unable to identify similar ones addressing the patterns' impact over more than one quality characteristics like in our case. For addressing the *internal validity*, the question ordering applied was for demographic data, the usability of design patterns, software quality characteristics related to design patterns, and ways of searching for patterns. We order to decrease the likelihood of bias arising from subsequent pattern rating questions influenced by the order of the patterns were listed and evaluated; we applied a matrix format for the rating questions ensuring that all question elements were presented together on the screen. In this way, the respondents were able to fill in their ratings in the order they did prefer and potential *rating fatigue* effects were avoided.

Regarding the way of administrating the survey, our main concern was whether the actual population sampled in our study can be regarded as a valid representative subset of the target population of software professionals in Bulgaria. First, the

sampling mechanism that we employed ensured random choice of respondents thanks to sending invitations to all the workers in many software companies and having the freedom to participate in the survey or not. According to Eurostat, the size of the target population in the year 2017 was 71,000. Therefore, for a 95% confidence level (i.e., for a 5% chance of our sample results differing from the true population average), 82 respondents result in a margin of error equal to 0.1082, i.e., the pattern ratings of the actual population may vary by ±10.82%. Such a maximal error could change only the order of patterns are rated according to a specific issue but not the patterns themselves. On the other hand, we were able to identify the gender ratio of the target population (Section 4.1), which appeared to differ at only 0.78% compared to the same of the actual population.

Finally, the analysis of the data received concerns the validity of the outcomes, which could be affected by *data validation, data coding,* and *consistency of the answers* [25]. In our survey, all the incomplete responses were removed so the quantitative analysis was conducted with responses to closed questions contained valid data. A formal coding of the raw quantitative data to the five-point Likert scale was applied. For rating the pattern usability, the five-point Likert scale was reduced to a three-point scale. We consider both the coding and scale reduction unlikely to have been significant. The third issue – consistency of the answers – cannot be estimated here, because we do not have multiple rating questions addressing the same concept and, therefore, we cannot assess reliability by statistics like Cronbach's *alpha*.

6. Conclusion

As in many other disciplines, software design applies knowledge structures representing "generic concepts stored in memory" known as "knowledge schema" [26]. Design patterns externalize the schemas of experienced designers and offer that knowledge to the others [5]. The article presents some of the results of a quantitative study aimed at determining the usability of object-oriented design patterns [1] in terms of achieving a better software quality, reported by software professionals in Bulgaria. It proves the fact that *software workers having different age, education, profession, and experience apply object-oriented design patterns for improving software quality characteristics such as testability, maintainability, extendibility, readability, reliability, and performance efficiency*. Although many of the young and not graduated software developers have no familiarity with design patterns, the majority of the professionals with intermediate or higher experience demonstrate good knowledge and reasonable and responsible attitude regarding advantages and shortcomings when applying patterns. Their responses reveal that Singleton (C), Factory Method (C), and Iterator (B) are both most recognized and useful, while Memento (B) and Flyweight (S) are both most unrecognized and useless. For patterns like Singleton (C) and Abstract Factory (C), the opinions are highly polarized. Unfortunately, the majority of the respondents much better utilize patterns for an optimized creation of objects and structures than for flexible representation of both the inter-object relationships and behavior. Most people surveyed apply design

templates mostly to create objects. The behavioral pattern group appears to be the most unpopular and rarely used, although having the number of patterns. On the other hand, the respondents with experience in patterns believe they bring essential improvements in software quality, though to a varying degree on quality characteristics depending on each pattern.

The presented over results provide a basis for further reflection on the reasons for the current situation of using design patterns. The fact that about 13% of the professionals with higher education have never used patterns, even in any student project, shows that as a whole education at universities does not focus enough on studying design patterns, though the correlation between the experience in object-oriented programming and in design patterns. If this topic would be affected during higher education, it will positively contribute to the skills and employability of new graduate students.

References

1. Gamma, E., R. Helm, R. Johnson, J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Reading, Addison-Wesley, 1995.
2. Riehle, D., H. Züllighoven. Understanding and Using Patterns in Software Development. – Theory and Practice of Object Systems, Vol. **2**, 1996, No 1, pp. 3-13.
3. Kohls, C., K. Scheiter. The Relation between Design Patterns and Schema Theory. – In: Proc. of 15th Conference on Pattern Languages of Programs (PLoP'08), ACM Press, 2008, pp. 1-14.
4. Schmidt, D. C., M. Stal, H. Rohnert, F. Buschmann. Pattern-Oriented Software Architecture. Patterns for Concurrent and Networked Objects. Vol. **2**. John Wiley & Sons, 2013.
5. Shalloway, A., J. R. Trott. Design Patterns Explained: A New Perspective on Object-Oriented Design. 2nd Ed. Pearson Education, India, 2005.
6. Boehm, B. W., K. J. Sullivan. Software Economics: A Roadmap. – In: Proc. of Conference on the Future of Software Engineering, ACM, 2000, pp. 319-343.
7. McNatt, W. B., J. M. Bieman. Coupling of Design Patterns: Common Practices and their Benefits. – In: Proc. of 25th Ann. Int. Computer Software and Applications Conf. (COMPSAC'01), IEEE, 2001, pp. 574-579.
8. Wendorff, P. Assessment of Design Patterns during Software Reengineering: Lessons Learned from a Large Commercial Project. – In: Proc. of 5th European Conference on Software Maintenance and Reengineering, IEEE, 2001, pp. 77-84.
9. Bieman, J. M., D. Jain, H. J. Yang. OO Design Patterns, Design Structure, and Program Changes: An Industrial Case Study. – In: Proc. of IEEE International Conference on Software Maintenance, IEEE, 2001, pp. 580-589.
10. Khomh, F., Y. G. Gueheneuce. Do Design Patterns Impact Software Quality Positively? – In: Proc. of 12th European Conference on Software Maintenance and Reengineering (CSMR'08), IEEE, 2008, pp. 274-278.
11. Hegedűs, P., D. Bán, R. Ferenc, T. Gyimóthy. Myth or Reality? Analyzing the Effect of Design Patterns on Software Maintainability. – Computer Applications for Software Engineering, Disaster Recovery, and Business Continuity, Springer, Berlin, Heidelberg, 2012, pp. 138-145.
12. Zhang, C., D. Budgen. A Survey of Experienced User Perceptions about Software Design Patterns. – Information and Software Technology, Vol. **55**, 2013, No 5, pp. 822-835.
13. Alghamdi, F. M., M. R. J. Qureshi. Impact of Design Patterns on Software Maintainability. – International Journal of Intelligent Systems and Applications, Vol. **6**, 2014, No 10, 41.
14. Questers. IT Industry Report. Bulgaria. Questers Press, January 2018.

15. M e y e r, B. Object-Oriented Software Construction. Vol. **2**. New York, Prentice Hall, 1988.
16. S o m m e r v i l l e, I. Software Engineering (International Computer Science Series). Addison Wesley, 2004.
17. A l e x a n d e r, C. A Pattern Language: Towns, Buildings, Construction. Oxford Univ. Press, 1977.
18. F e n t o n, N., J. B i e m a n. Software Metrics: A Rigorous and Practical Approach. CRC Press, 2014.
19. A L-m s i e'd e e n, R. F. Visualizing Object-Oriented Software for Understanding and Documentation. – International Journal of Comp. Science and Inf. Security, Vol. **13**, 2015, No 5, pp. 18-27.
20. V l i s s i d e s, J., J. C o p l i e n, N. K e r t h. Pattern Languages of Program Design. Addison-Wesley Professional, 1996.
21. W y d a e g h e, B., K. V e r s c h a e v e, B. M i c h i e l s, I. V a n B a m m e, E. A r c k e n s, V. J o n c k e r s. Building an OMT-Editor Using Design Patterns: An Experience Report. – Proc. of Technology of Object-Oriented Languages, IEEE, 1998, pp. 20-32.
22. T a h v i l d a r i, L., K. K o n t o g i a n n i s. On the Role of Design Patterns in Quality-Driven Re-Engineering. – In: Proc. of 6th European Conference on Software Maintenance and Reengineering, IEEE, 2002, pp. 230-240.
23. Z h a n g, C., D. B u d g e n. What do We Know about the Effectiveness of Software Design Patterns? – IEEE Transactions on Software Engineering, 2012, No 38, pp. 1213-1231.
24. K i t c h e n h a m, B. A., S. L. P f l e e g e r. Principles of Survey Research. Part 4: Questionnaire Evaluation. – ACM Software Engineering Notes, 2002, No 27, pp. 20-23.
25. K i t c h e n h a m, B. A., S. L. P f l e e g e r. Principles of Survey Research. Part 6: Data Analysis. – ACM Software Engineering Notes, 2003, No 28, pp. 24-27.
26. K o h l s, C., K. S c h e i t e r. The Relation between Design Patterns and Schema Theory. – In: Proc. of 15th Conference on Pattern Languages of Programs (PLoP'08), ACM Press, 2008, pp. 1-14.

Received: 29.07.2020; Second Version: 31.10.2020; Accepted: 06.11.2020