Identifying Software Complexity Topics with Latent Dirichlet Allocation on Design Patterns

Sabina-Cristiana NECULA, Cătălin STRÎMBEI
Alexandru Ioan Cuza University of Iasi, Iasi, Romania;
sabina.necula@uaic.ro, linus@uaic.ro

The scientific literature has paid limited attention to studying software complexity subjects from the design point of view. There is a significant number of papers that study software complexity in relation with maintenance, refactoring, source code changes and that establish metrics for measuring software complexity. This paper compares design patterns and software complexity in order to identify trends of research in the software complexity area. For this purpose, we assess the strengths and weaknesses of software complexity scientific articles through the lens of design patterns. We have reviewed 1068 papers via latent Dirichlet allocation technique (LDA) for our work. We found that existing software complexity paths disproportionate emphasis in how software complexity could benefit from design patterns instead on how contributions to design patterns can benefit from software complexity.

Keywords: *latent Dirichlet allocation, design patterns, software complexity, software complexity topic modelling*

1 Introduction

Software complexity is a key property in software engineering and developing applications. The subject relates to refactoring, reusability, reducing software management project costs, and large infrastructures at low cost. Although there are many software metrics, the scientists still consider there are subjects to improve by scientific studies. Because the modern software engineering relates to object oriented field when designing applications and because of the wide area, that software complexity supposes, we decided to study the subject of software complexity by approaching the field of design patterns, hopping to identify the main topics to study and future trends in research.

Various topics on object oriented design have been proposed over the years. Design patterns are a subject of interest because they offer solutions for the coupling and cohesion between different layers of an application. The problems due to of a design with high coupling are: changes in related classes force local changes; harder to understand in isolation; harder to reuse because it requires additional presence of other classes. The problems due to a design with low cohesion are: hard to understand; hard to reuse or to maintain. High cohesion means that a class has moderate responsibility in one functional area and it collaborates with other classes to fulfill a task. Software complexity can be reduced by designing systems with the weakest possible coupling between modules [1].

Historically, complexity in programs arising because of the number of conditional and iterative statements has been measured using the cyclomatic complexity metric [2]. Refactoring code with design patterns reduces complexity, although it increases the number of classes [3]. The authors show that design patterns do not always improve the quality of systems. Some patterns are reported to decrease some quality attributes and to not necessarily promote reusability, expandability, and understandability. Also, they bring further evidence that design patterns should be used with caution during development because they may actually impede maintenance and evolution. Their study also reveals that object-oriented principles may not be so "good" as they may not necessarily result in systems with good quality.

However, we consider that the subject of studying the effect that design patterns might have on software complexity is not very well represented. The scope of this paper is to identify the main topics, the trends, and to test if there is a correlation between the two subjects.

DOI: 10.12948/issn14531305/23.4.2019.01

2 Materials and methods

In this section, we present the research goals and questions to be answered, and we describe the inclusion or selection criteria for the studies chosen to analyze and data collection.

The purpose of this article is to get a broad and current overview of the two subjects considered in this paper: design patterns and software complexity. The analysis was realized on academic journal articles. The search for papers was conducted in 2019 on Thomson Reuters' Web of Science, which has a large interdisciplinary database of academic texts, and limited to peer reviewed articles and reviews in English.

We realized two searches in the title, abstract, and keywords of papers from ISI Clarivate:

- design patterns, which returned 2045 articles;
- "software complexity", which returned 302 articles.

Data selection is presented in Figure 1.

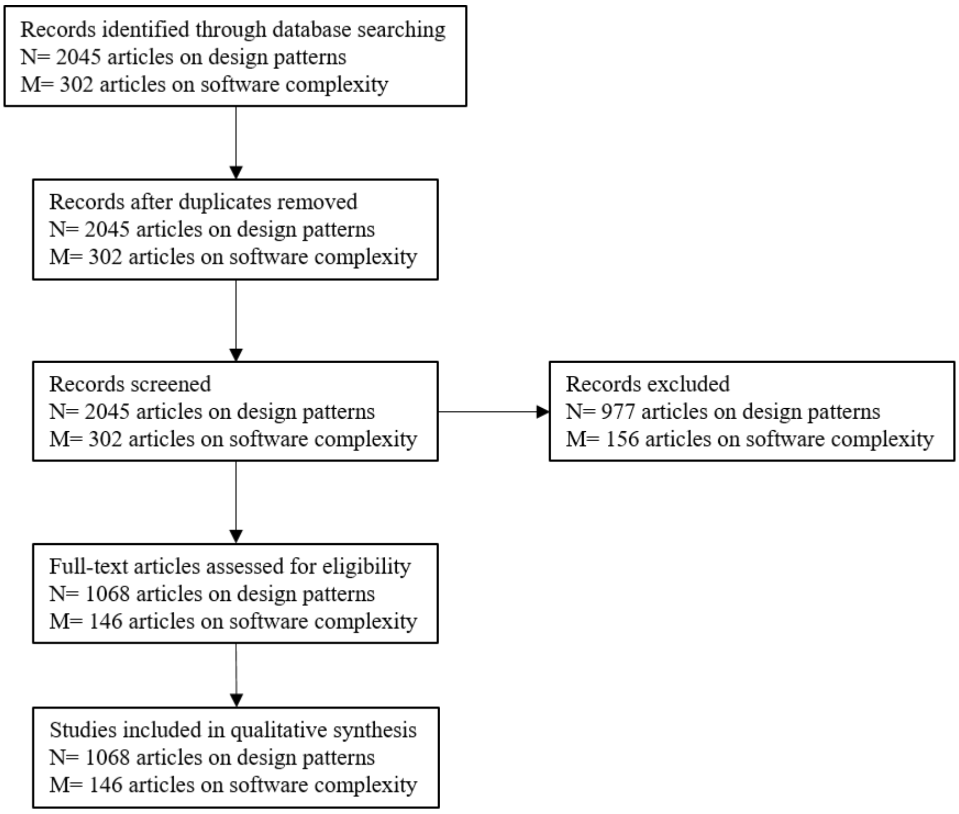

Records identified through database searching
N= 2045 articles on design patterns
M= 302 articles on software complexity
Records after duplicates removed
N= 2045 articles on design patterns
M= 302 articles on software complexity
Records screened
N= 2045 articles on design patterns
M= 302 articles on software complexity
Records excluded
N= 977 articles on design patterns
M= 156 articles on software complexity
Full-text articles assessed for eligibility
N= 1068 articles on design patterns
M= 146 articles on software complexity
Studies included in qualitative synthesis
N= 1068 articles on design patterns
M= 146 articles on software complexity

Fig. 1. Papers dataset

In order to identify the topics from software complexity subject area we applied LDA on the corpus of articles belonging to the design patterns subject. For the LDA analysis, we used the abstracts of the selected papers. Figure 2 presents the approach of our study.

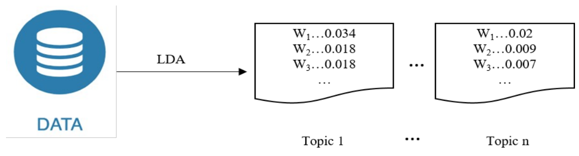


Fig. 2. LDA research approach

The abstracts are expected to give a sufficient indication of what is the subject of the paper and thus provide an overview of the topics discussed in the respective fields [4]. In natural language processing, latent Dirichlet allocation (LDA) is a generative statistical model that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar. LDA is an example of a topic model. The method for topic modelling employed in this paper, Latent Dirichlet Allocation (LDA), has greater flexibility using as input the whole abstracts of the papers. LDA is a generative

probabilistic topic model proposed by [4], which can be used for the unsupervised identification of underlying topics in a large corpus of data without any prior knowledge of the topics [5-6]. Although the documents, or abstracts, are known and observed, the topics are hidden or latent [7].
The total number of abstracts (papers) is noted with N. For each abstract d belonging to N, we extracted a vector of words Xd=[Xd1, Xd2, …, XdWd] where Wd is the number of words in abstract d. W is the number of unique words in the dataset, and and V=[w1,w2,...,wn] is the vocabulary of words. Rather than representing a text T in its feature space as {Word_i: count(Word_i, T) for Word_i in V}, we can represent the text in its topic space as {Topic_i: weight(Topic_i, T) for Topic_i in Topics}. The LDA topic model algorithm requires a document word matrix as the main input, e.g. Document-Word Matrix (or Document-Term Matrix). DWM[i][j] = The number of occurrences of word_j in document_i.
Topics are latent variables composed of word distributions. Producing an interpretable solution is the beginning, not the end, of an analysis. To draw adequate conclusions, the interpretation of the latent variables must be substantially validated [8]. Several authors proposed guidance for evaluation and validating LDA models [9]. We studied the coherence and perplexity of different LDA resulted models, choose the model that had the best coherence value, filtered the articles written on software complexity starting from design patterns discovered topics and conducted the analysis of resulted papers by reviewing their subject and method of research. The LDA was carried out in Python. We used LDAvis to visualize and optimize the number of topics [10].
The selection criteria for our sample of studies were based on the following considerations: 1) The scientific articles indicate the concerns from the research field; 2) The scientific articles published in the main journals from the ISI Web of Science database offer a broad overview on the subject.
This work is based on the three goals with related motivations presented in Table 1.

- G1: to investigate the relation between design patterns subject and software complexity by identifying the proportion between the articles written on the subject of design patterns and software complexity and test the correlation.
- G2: to investigate if the two subjects are statistically different.
- G3: to investigate the topics on design patterns from the software complexity subject.

Research questions were derived from each goal, and testable hypotheses formulated, as summarized in Table 1.

Table 1. Research goals and questions

Goal	Research question	Motivation	Null Hypothesis H0
G1	Q1: Is there a directional relationship between the topics on the subject of design patterns and the topics on the subject of software complexity?	The subject of design patterns and the subject of software complexity belong to the same field of study, namely software engineering	No linear relationship between the two subjects
G2	Q2: Are the two subjects of study different?	While it has been identified that design patterns are important for software complexity, the topics on the subject vary and treat different aspects	The two subjects do not differ
G3	Q3: What are the topics studied on the subject of design patterns from software complexity "world"		

The enunciated hypothesis that we established in our study were:

H1: there is a linear relationship between the subject of design patterns and the articles written on the subject of software complexity
H2: there is a difference in the topics treated by the articles belonging to these 2 subjects.
With respect to related work, our study intends to be an attempt to evaluate the relationship between the subject of design pattern and software complexity. Also, it is the first work that examines the topics from software complexity field of research with techniques from natural language processing.
Additionally, we outlined the steps performed in the methodology: 1) identifying the articles written on the two subjects, identifying the topics in the field of design patterns by using LDA; 2) identifying the proportion of articles per each topic from design patterns and software complexity subject; 3) testing the hypotheses. Establishing whether there is a relationship between the two subjects has several applications in software engineering, including: A1) Predictions of topics on the design pattern subject; A2) Predictions of topics on design pattern subject across software complexity subject.

3 Results

The proportion over Web of science categories for the papers with the design patterns subject is presented in Table 2.

Table 2. The number of articles from the design patterns subject on science categories (Top Ten)

Web of Science category	Number of articles	% from Total number of articles (2045)
Computer Science	1068	52.22%
Computer Science; Engineering	147	7.19%
Engineering	62	3.03%
Education & Educational Research	45	2.20%
Computer Science; Telecommunications	35	1.71%
Computer Science; Engineering; Telecommunications	24	1.17%
Science & Technology - Other Topics	22	1.08%
Engineering; Telecommunications	21	1.03%
Computer Science; Engineering; Operations Research & Management Science	19	0.93%
Business & Economics	16	0.78%

We chose to identify the topics starting from the Computer Science category because this category is well represented by a high number of articles. The software complexity subject is represented by a number of 302 articles. The distribution across different science categories is presented in Table 3.
The first step in the pre-processing was to remove stop-words. The stop-words are words such as ”the”, ”a”, ”I”, ”him”, etc. Next, we created bi-grams and tri-grams. These terms are new words that are combinations of words that are commonly juxtaposed. Next, we lemmatized the words. This involved removing inflectional endings, thus returning it to its base form. An example is changing the word ”Working” to ”Work”. This helps with topic modeling and interpretation. We applied LDA on the abstracts from the design patterns (computer science) articles.

Table 3. The software complexity across science categories (Top Ten)

Science category	Number of articles	% from Total number of articles (302)
Computer Science	146	48.34%
Computer Science; Engineering	42	13.91%
Engineering	19	6.29%
Engineering; Telecommunications	9	2.98%
Mathematics	6	1.99%
Computer Science; Engineering; Telecommunications	5	1.66%
Computer Science; Engineering; Operations Research & Management Science	4	1.32%
Computer Science; Operations Research & Management Science	4	1.32%
Science & Technology - Other Topics	3	0.99%
Automation & Control Systems; Computer Science	3	0.99%

The topics, the first 30 terms and their graphical visualization are presented in Fig. 3a and Fig. 3b for the first and the second topic, respectively. The topics are circles in the two-dimensional plane whose centers are determined by computing the distance between topics [11]. The overall topic prevalence is represented by the areas of the circles, where the topics are sorted in the decreasing order of prevalence. In the right part of the figure, a pair of overlaid bars represent both the corpus-wide frequency of a given term as well as the topic-specific frequency of the term, as in [12].

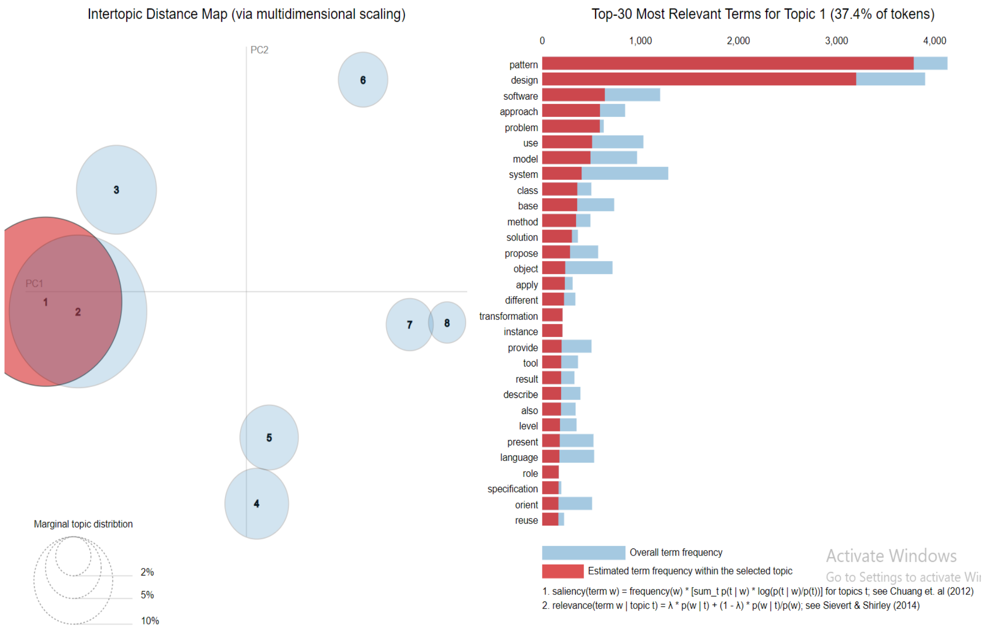


Fig. 3a. Intertopic distance map via multidimensional scaling considering the marginal topic distribution employing the first and second principal components - Topic 1(PC1 and PC2) (LDAvis) - detailed further in Table 4

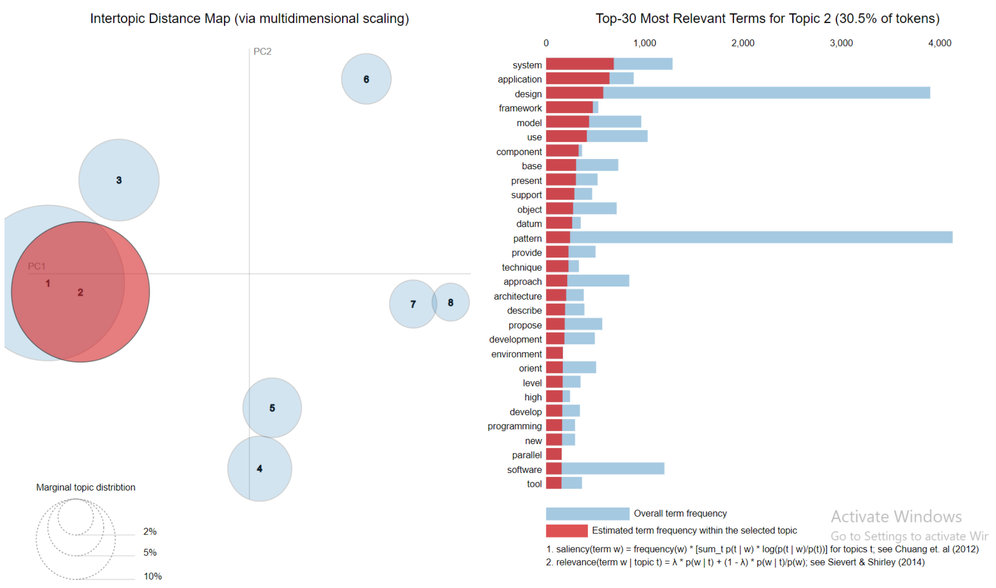


Fig. 3b. Intertopic distance map via multidimensional scaling considering the marginal topic distribution employing the first and second principal components - Topic 2 (PC1 and PC2) (LDAvis) - detailed further in Table 4

The top 30 most salient terms from the entire dataset (when no topic is selected) are: pattern, design, software, language, code, ontology, object, class, aspect, method, application, implementation, service, programming, architecture, framework, process, type, system, development, program, problem, orient, performance, web, quality, mechanism, cloud, structure, component.

We optimized the model by identifying the number of topics that provide the best coherence. It seems that the optimum number of topics from the subject area of design patterns is 8. The coherence measure is equal to 0.7612.

The resulted topics from the design patterns subject of research and their words and probability to appear in relation with each other are presented in Table 4.

Table 4. Topics on design patterns (computer science)- LDA

Topic	Words/ Probability to appear in the same document							
Model development	framew'	0.042	component	0.034	applic'	0.022	reuse'	0.015
System model	model'	0.061	specific'	0.016	composit'	0.011	role'	0.011
Analysis and program	support'	0.027	Tool	0.026	model'	0.023	analysis'	0.018
Method implementation	perform'	0.018	technique'	0.017	develop'	0.009	feature'	0.009
Software code	code'	0.027	method'	0.021	qualiti'	0.016	software'	0.015
Software language	implem'	0.023	languag'	0.050	program'	0.026	code'	0.016
System software	softwar'	0.061	problem'	0.049	system'	0.015	develop’	0.012
Application development	system'	0.088	applic'	0.055	develop'	0.031	requir'	0.014

We extracted the papers belonging to design patterns and software complexity on each topic. The results are presented in Table 5. The analyze revealed that there is an important number of articles written on the subject of software complexity that benefit from the well-grounded subject of design patterns, but the numbers of articles on design patterns that benefit from software complexity research articles is not that well represented.

Table 5. The statistical results

Topic	Articles written on the subject of design patterns (Total)	Articles written on the subject of software complexity	Design patterns as proportion from total (1068 articles)	Software complexity as proportion from total (146 articles)	Software complexity versus design patterns (as ratio)
Model development	98	32	9.18%	21.77%	2.37
System model	101	36	9.46%	24.49%	2.59
Analysis and program	22	19	2.06%	12.93%	6.27
Method implementation	31	6	2.90%	4.08%	1.41
Software code	65	38	6.09%	25.85%	4.25
Software language	51	17	4.78%	11.56%	2.42
System software	128	69	11.99%	46.94%	3.92
Application development	119	22	11.14%	14.97%	1.34

Pearson correlation= 0.72 t-test=0.005, p value=0.01

We found that between design patterns subject and software complexity subject there is a direct and strong relationship (Pearson correlation=0.72), so we reject the null hypothesis. The t-test value was 0.005 for a p value of 0.01, so we were able to reject the null hypothesis. Therefore, there is a significant difference in the way the two subjects are treated by the authors of scientific articles from the mainstream research. We continued our analysis towards identifying the specific topics on design patterns from the software complexity subject. We eliminated the common keywords: design, system, complexity, software, application, model, program. Therefore, we were able to observe the topics that relate the two subjects. We computed the most representative article for each topic, both for design patterns subject of research and for software complexity subject of research. Table 6 presents the title of the papers per each topic.

Table 6. The articles that contributed most to topic identification

Topic	**Topic keywords**	**Topic Contribution (percentage)**	**Title**
1	object, language, orient, implementation, programming, structure, type, class, java, support, code	0.5542	Software complexity measurement
2	framework, process, development, propose, research, application, case, base, present, develop, reuse, component	0.582	An architecture design process using a supportable meta-architecture and round trip engineering

3	code, method, technique, result, set, source, program, perform, feature	0.5975	Eliminating synchronization faults in air traffic control software via design for verification with concurrency controllers
4	pattern, model, design, approach, describe, tool, transformation, specification, apply, define, composit	0.5743	Assessing maintainability change over multiple software releases
5	pattern, design, problem, software, solution, class, study, quality, reuse, experience, code, method	0.6946	Empirical assessment of design patterns' fault-proneness at different granularity levels
6	system, application, component, architecture, performance, parallel, distribute, time, base, develop, requir	0.5941	The design and performance of a pluggable protocols framework for real-time distributed object computing middleware
7	design, pattern, software, system, approach, base, analysis, propose, change, instance, problem	0.5835	UML design pattern metamodel-level constraints for the maintenance of software evolution
8	datum, domain, ontology,tool, model, service, application, web, knowledge, information, user, specific, support	0.6193	The GeoVoCamp Workshop Experience and Ontology Design Pattern Development

In general, the articles written on the subject of software complexity address the main topic of "metric". LDA returned the word metric in each identified topic. Usually, these metrics are well established in theory and in practice. We noticed one single paper, entitled *How changes affect software entropy: an empirical study* [13], in which the authors analyzed how changes affect software entropy by: the presence of refactoring activities, the number of developers working on a source code file, the participation of classes in design patterns, and the different kinds of changes occurring on the system, classified in terms of their topics extracted from commit notes. The research subject of design patterns is represented by topics like pattern development and pattern usage.

The distribution of papers per each subject, namely design patterns and software complexity, for each topic starting from topic 1(T1) till topic 8 (t8) is presented in Figure 5.

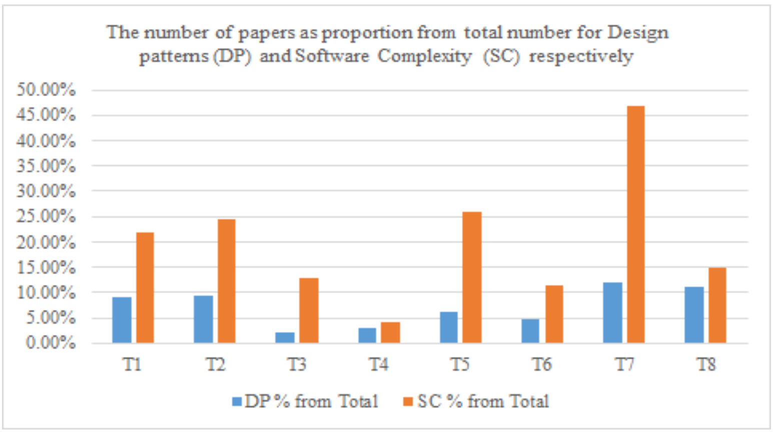


Fig. 5. The number of papers as proportion from total number for Design patterns (DP) and Software Complexity (SC) respectively

We chose to analyze in the Discussion section the first three topics belonging to software complexity that have the biggest proportion over design patterns subject: analyzing software complexity, source code change, system software complexity.

4. Discussions

Complexity is an important property to analyze when developing software. Software measures are the way to quantify the structural complexity of software. The analyzing software complexity topic contains papers that discuss developing tools and papers dedicated to developing metrics.

The topic of code changes is analyzed intensively by the scientific literature. The interests are on sustainability [14], reusability, maintenance, decreasing complexity. It is interesting to observe that there is a significant number of papers which approach the topic by using machine learning techniques. The main subject is version to version code change. The topic of source code change is approached empirical through case studies or analysis on open source projects. As software changes requires some form of managing the changes, some authors proposed the use of repositories. They consider that the company must have a central knowledge repository with software specifications [15-17], designs and code from previous system developments. The central knowledge base can be used through Case-Based Reasoning.

An important number of authors treat the problem of maintenance. They hypothesize that source code complexity exerts a causal influence on maintenance difficulty experienced during the system test phase of the product [18]. At least three components contribute to the complexity of the software maintenance effort: (1) the code and documentation being produced, (2) the process used to manage the maintenance, and (3) the maintenance and target computer system environments [19].

The authors that study on the topic of system software complexity approach subjects like software metrics or refactoring but in the context of software engineering or re-engineering and systems development. Also, the subject of software architecture design process is approached in [20] where a supportable meta-architecture (SMA) and roundtrip engineering is proposed for large software projects.

There are authors who study requirements engineering in relation to software complexity [21]. They developed a quality-driven RE framework and tool that applies knowledge management techniques and quality ontologies to support RE activities. Software refactoring is an important subject to study when designing systems. Alkhalid et.al [22] use clustering as a pattern recognition technique to assist in software refactoring activities at the package level. The approach presents a computer aided support for identifying ill-structured packages and provides suggestions for software designer to balance between intra-package cohesion and inter-package coupling. A comparative study is conducted applying three different clustering techniques on different software systems. In addition, the application of refactoring at the package level using an adaptive k-nearest neighbor (A-KNN) algorithm is introduced.

Artificial intelligence techniques are used in detecting program modules having high risk in the maintenance phase. There are authors who developed a neural network model to classify program modules as either high or low risk based on multiple criterion variables. The inputs to the model included a selection of software complexity metrics collected from a telecommunications system. Applications of intelligent software systems are proliferating. As these systems proliferate, understanding and measuring their complexity becomes vital, especially in safety-critical environments. The results suggest that users significantly prefer simple decision support and user interfaces, even when sophisticated user interfaces and complex decision support capabilities have been embedded in the system [23].

Future trends or newcomers

Software complexity represents a large research field that still searches its innovative ideas. It seems that cognitive software complexity topic is gaining success in the last years, as is studying the software metrics in real time, in the context of embedded systems.

Software entropy, maintenance, and refactoring remain central preoccupations but their analysis is approached lately with artificial intelligence techniques.
Among the papers written on the subject of software complexity, we analyzed the papers that were addressed with artificial intelligence techniques. The subjects and the research method used in the respective paper are presented in Table 7.

Table 7. Subjects from software complexity subject related to design patterns

Subject	The research method used in the paper
cohesion, estimation	clustering as pattern recognition method to assist in software refactoring
quality requirements engineering	knowledge management techniques and quality ontologies to support Requirements Engineering (RE) activities
software reusability	case based reasoning
context IT, software maintenance	the cognitive complexity metric as a measure of version to version source code change.
software change prediction	neural network based on software complexity measurements
software reusability	WordNet (vocabulary) for case-based retrieval
software maintenance	entropy semantically-based metric

According to the Forrester Research report on AI's impact on software development [24], the bulk of the interest in applying AI to software development lies in automated testing and bug detection tools. The article 6 ways AI transforms how we develop software [25] discusses rapid prototyping, intelligent programming assistants, automatic analytics & error handling, automatic code refactoring, precise estimates, and strategic decision-making. Our results confirm this idea.

5 Conclusions

In reviewing the literature, no data was found on the association between design patterns and software complexity or in identifying software complexity topics. The current study found that although software complexity and design patterns belong to the same subject area, namely software engineering, the topics vary. The most interesting finding was that there is possible to identify topics on software complexity by identifying topics on design patterns. Another important finding was that measuring software complexity and evaluating its effects on the developed systems is approached very often with artificial intelligence techniques. This combination of findings provides some support for the conceptual premise that design patterns might be studied from the software complexity point of view.
In conclusion, in this study, the relations between software complexity associated metrics and design patterns have been investigated. Also, this study emphasizes the importance of design patterns, the lack of standard metrics for design patterns, and the lack of standard ways for studying design patterns in relation with software complexity. The LDA technique proved its reliability in studying topics from the field of software complexity.

References

[1] B. Henderson-Sellers, *A book of object-oriented knowledge: an introduction to object-oriented software engineering*. Prentice-Hall, Inc., 1996.
[2] A.H. Watson, D.R. Wallace, and T.J. McCabe, *Structured testing: A testing methodology using the cyclomatic com-*

plexity metric (Vol. 500, No. 235). US Department of Commerce, Technology Administration, National Institute of Standards and Technology, 1996.

[3] F. Khomh and Y.G. Gueheneuce, Do design patterns impact software quality positively?. In 2008 12th *European Conference on Software Maintenance and Reengineering* (pp. 274-278), IEEE, 2008.

[4] D.M. Blei, T.L. Griffiths, and M.I. Jordan, The nested chinese restaurant process and bayesian nonparametric inference of topic hierarchies. *Journal of the ACM (JACM)*. 57(2), p.7, 2010.

[5] J. Guerreiro, J., P. Rita, and D. Trigueiros, A text mining-based review of cause-related marketing literature, *Journal of Business Ethics*, 2016, 139(1), pp.111-128.;

[6] C.R. Sugimoto et. al, The shifting sands of disciplinary development: Analyzing North American Library and Information Science dissertations using latent Dirichlet allocation. *Journal of the American Society for Information Science and Technology*, 2011, 62(1), pp.185-204.

[7] A. Piepenbrink and E. Nurmammadov, Topics in the literature of transition economies and emerging markets, *Scientometrics*, 2015, 102(3), pp.2107-2130.

[8] P. DiMaggio, M. Nag and D. Blei, Exploiting affinities between topic modeling and the sociological perspective on culture: Application to newspaper coverage of US government arts funding, *Poetics*, 2013, 41(6), pp.570-606.

[9] D. Maier et.al, Applying LDA topic modeling in communication research: Toward a valid and reliable methodology, *Communication Methods and Measures*, 2018, 12(2-3), pp.93-118.

[10] C. Sievert and K. Shirley, LDAvis: A method for visualizing and interpreting topics. In *Proceedings of the workshop on interactive language learning, visualization, and interfaces*, 2014, (pp. 63-70).

[11] J. Chuang, D. Ramage, C. Manning and J. Heer, Interpretation and trust: Designing model-driven visualizations for text analysis. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2012, (pp. 443-452). ACM.

[12] J. Chuang, C.D. Manning and J. Heer, Termite: Visualization techniques for assessing textual topic models. In *Proceedings of the international working conference on advanced visual interfaces*, 2012, (pp. 74-77). ACM.

[13] G. Canfora, L. Cerulo, M. Cimitile and M. Di Penta, How changes affect software entropy: an empirical study, *Empirical Software Engineering*, 2014, 19(1), pp.1-38.

[14] C.C. Venters et.al, Software sustainability: Research and practice from a software architecture viewpoint, *Journal of Systems and Software*, 2018, 138, pp.174-188.

[15] L. Hurbean and D. Fotache, The challenge of enterprise systems: harmonization of ERP systems with business processes. *Scientific Annals of the'Alexandru Ioan Cuza'University, Economic Sciences Series*, 2010.

[16] V.D. Păvăloaia, M.R. Georgescu, D. Popescul, D. and L.D. Radu, ESD for public administration: An essential challenge for inventing the future of our society, Sustainability, 2019, 11(3), p.880.

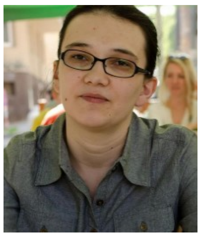
[17] F. Dumitriu, D. Oprea, G. Mesnita, N. Gavriluta, R. Raducanu, and M. Iliescu, Issues and strategy for agile global software development adoption. In Proceeding of the 3rd World Multiconference on Applied Economics, Business and Development, AEBD. (Vol. 11, pp. 1-3), 2011, WSEAS Press.

[18] D.L. Lanning and T.M. Khoshgoftaar, Modeling the relationship between source code complexity and maintenance difficulty, Computer, 1994, 27(9), pp.35-40.

[19] G.E. Stark and P. Oman, A survey instrument for understanding the complexity of software maintenance. Journal of Software Maintenance: Research and Practice, 1995, 7(6), pp.421-441.

[20] H. Gümüşkaya, An architecture design process using a supportable meta-architecture and roundtrip engineering. In International Conference on Advances in Information Systems, 2006, (pp. 324-333).

Springer, Berlin, Heidelberg.
[21] T.H. Al Balushi, P.R.F. Sampaio and P. Loucopoulos, Eliciting and prioritizing quality requirements supported by ontologies: a case study using the E licit O framework and tool. Expert Systems, 2013, 30(2), pp.129-151.
[22] A. Alkhalid, M. Alshayeb and S.A. Mahmoud, Software refactoring at the package level using clustering techniques. *IET software*, 2011, 5(3), pp.274-286.
[23] E. Coskun and M. Grabowski, Software complexity and its impacts in embedded intelligent real-time systems, *Journal of Systems and Software*, 2005, 78(2), pp.128-145.
[24] D.L. Giudice, C. Mines, A. LeClair, R. Curran, A. Homan, How AI Will Change Software Development And Applications, Forrester report 2016, available at https://reprints.forrester.com/#/assets/2/108/'RES121339'/reports
[25] M. Yao, 6 Ways AI Transforms How We Develop Software, Forbes 2018, available at https://www.forbes.com/sites/mariyayao/2018/04/18/6-ways-ai-transforms-how-we-develop-software/


Sabina-Cristiana NECULA (b. July 27, 1979) received her PhD in Accounting (Business In-formation Systems) (2007). She is currently a Scientific Researcher at Alexandru Ioan Cuza University of Iasi, Faculty of Economics and Business Administration. Her current research interests include Semantic Web standards and technologies, Decision Support Systems, Business Information Systems. She is the author of more than 30 scientific papers.

Cătălin STRÎMBEI has graduated the Faculty of Economics and Business Administration of Alexandru Ioan Cuza University of Iaşi in 1997. He holds a PhD diploma in Cybernetics, Statistics and Business Informatics from 2006 and he has joined the staff of the Faculty of Economics and Business Administration as teaching assistant in 1998 and as associate professor in 2013. Currently he is teaching Object Oriented Programming, Multi-Tier Software Application Development and Database Design and Administration within the Department of Business Information Systems, Faculty of Economics and Business Administration, Alexandru Ioan Cuza University of Iaşi. He is the author and co-author of four books and over 30 journal articles in the field of object oriented development of business applications, databases and object oriented software engineering.