DOI: 10.15514/ISPRAS-2021-33(3)-1

What Software Architecture Styles are Popular?

A.A Mitsyuk, ORCID: 0000-0003-2352-3384<amitsyuk@hse.ru>
N.A. Jamgaryan, ORCID: 0000-0001-9964-5850 <nazhamgaryan@edu.hse.ru>
HSE University,
11, Pokrovsky boulevard, Moscow, 109028, Russia

Abstract. One can meet the software architecture style's notion in the software engineering literature. This notion is considered important in books on software architecture and university sources. However, many software developers are not so optimistic about it. It is not clear, whether this notion is just an academic concept, or is actually used in the software industry. In this paper, we measured industrial software developers' attitudes towards the concept of software architecture style. We also investigated the popularity of eleven concrete architecture styles. We applied two methods. A developers' survey was applied to estimate developers' overall attitude and define what the community thinks about the automatic recognition of software architecture styles. Automatic crawlers were applied to mine the open-source code from the GitHub platform. These crawlers identified style smells in repositories using the features we proposed for the architecture styles. We found that the notion of software architecture style is not just a concept of academics in universities. Many software developers apply this concept in their work. We formulated features for the eleven concrete software architecture styles and developed crawlers based on these features. The results of repository mining using the features showed which styles are popular among developers of open-source projects from commercial companies and non-commercial communities. Automatic mining results were additionally validated by the Github developers survey.

Keywords: software architecture style; software design; code smells; software repository mining; survey

For citation: Mitsyuk A.A., Jamgaryan N.A. What Software Architecture Styles are Popular? Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 3, 2021, pp. 7-26. DOI: 10.15514/ISPRAS-2021-33(3)-1

Acknowledgments. This work is an output of a research project implemented as part of the Basic Research Program at the Natinal Research University Higher School of Economics (HSE University). We thank our colleagues from PAIS Lab (HSE University) whose advice was very helpful in doing our developer surveys better. In particular, Sergey A. Shershakov proposed useful improvements.

Какие стили архитектуры программного обеспечения популярны?

А.А. Мицюк, ORCID: 0000-0003-2352-3384<amitsyuk@hse.ru>
Н.А. Жамгарян, ORCID: 0000-0001-9964-5850 <nazhamgaryan@edu.hse.ru>
Национальный исследовательский университет Высшая школа экономики,
109028, Россия, Москва, Покровский бульвар 11

Аннотация. В литературе по программной инженерии можно встретить понятие архитектурного стиля программного обеспечения (ПО). Во многих книгах по архитектуре ПО и академических лекциях это понятие рассматривается как одно из важных. Однако, многие разработчики-практики пессимистично настроены в отношении понятия архитектурного стиля. Таким образом, не вполне понятно, является ли данное понятие чисто академической концепцией или действительно используется разработчиками прикладного программного обеспечения. В этой статье делается попытка оценить отношение разработчиков-практиков к концепции архитектурного стиля ПО. Также оценивается популярность

одиннадцати конкретных архитектурных стилей. Применяются два метода. Опрос разработчиков был применен для оценки отношения разработчиков и определения того, считает ли сообщество разработчиков возможным автоматическое распознавание архитектурных стилей. Для интеллектуального анализа открытого исходного кода с платформы GitHub применялись автоматические скрипты. Эти скрипты позволяют выявлять факт использования стилей в конкретных репозиториях. Скрипты работают на основе самостоятельно разработанных наборов свойств для выбранных стилей. Было обнаружено, что понятие стиля архитектуры программного обеспечения – это не только «университетская» концепция. Многие разработчики ПО применяют это понятие и соответствующую концепцию в своей работе. В работе сформулированы свойства для одиннадцати архитектурных стилей ПО и описаны разработанные на основе этих свойств автоматические скрипты. Результаты интеллектуального анализа репозиториев с использованием предложенных свойств показали, какие стили популярны среди разработчиков проектов с открытым исходным кодом, опубликованных коммерческими компаниями и некоммерческими сообществами. Результаты интеллектуального анализа репозиториев дополнительно валидируются опросом GitHub-разработчиков.

Ключевые слова: стиль архитектуры программного обеспечения; проектирование программного обеспечения; запахи кода; интеллектуальный анализ репозиториев с исходным кодом; опрос

Для цитирования: Мицюк А.А., Жамгарян Н.А. Какие стили архитектуры программного обеспечения популярны? Труды ИСП РАН, том 33, вып. 3, 2021 г., стр. 7-26 (на английском языке). DOI: 10.15514/ISPRAS–2021–33(3)–1.

Благодарности. Работа выполнена в рамках Программы фундаментальных исследований НИУ ВШЭ. Мы благодарим наших коллег из PAIS Lab (НИУ ВШЭ), чьи советы помогли нам улучшить наши исследования разработчиков. В частности, полезные улучшения предложил С.А. Шершаков.

1. Introduction

Software architecture [1] is a discipline within software engineering dealing with software systems' structural and behavioral design. Software architects and designers define how the system is organized, its components, how these components communicate, etc. Software engineering literature (see, for example, foundational works by Shaw and Garlan [2], Taylor et al. [3], Richards and Neal [4]) applies a notion of *architecture style* or *pattern*. Shaw and Garlan [2] define it as follows: «*An architectural style defines: a family of systems in terms of a pattern of structural organization; a vocabulary of components and connectors, with constraints on how they can be combined*.» Taylor et al. [3] proposed another definition: «*An architectural style is a named collection of architectural design decisions that (1) are applicable in a given development context, (2) constrain architectural design decisions that are specific to a particular system within that context, and (3) elicit beneficial qualities in each resulting system*.» These definitions are general and relatively abstract as well as most other definitions from software engineering books and papers. Usually, no clues on how these styles can be identified and implemented in a concrete software source code are given.

This work summarizes our team's first results to understand better the concept of *software architecture style* and make it more tangible.

To do so, we first tried to find out the developers' attitude towards the concept of software architectural style. Are real non-academic developers familiar with this concept in general and with different particular styles? Do they consider this concept useful in their everyday professional activities? Secondly, we tried to identify empirical features of software architecture styles, which can be used in practice to recognize the usage of software architecture styles in Java and Python programs.

For our research, certain architecture styles were chosen. Then, we chose a small sample of software repositories, which were investigated to get empirical features of the architecture styles in source

code. Afterward, the crawlers were written and applied to parse the bigger sample of open source repositories on GitHub[1] service.

Besides, we conducted two developers' surveys. The first survey aimed to find out the developers' attitude towards the concept of software architecture styles. We held the survey to understand better whether this topic is worth researching. The second survey aimed to validate the results of the crawlers' parsing.

The aim of our research project – to identify empirical features of architecture styles – is new to software engineering, while the applied methods are well known. Surveys and repository mining were applied in many other research projects on code smells detection and design patterns identification (see Section 6). The methods we used had shown themselves as feasible in exploratory research projects.

Due to the first survey results, developers have *positive* attitude towards architecture styles. Many of them apply this concept in their projects, and even more of them think it is beneficial to be acquainted with the concept. In data provided by the automatic crawlers we found, how much each of the chosen architecture styles is used in practice. We validated the results of crawlers mining using the second survey.

2. Research Questions and Paper Structure

In this paper, we consider the three following research questions.

RQ1: *What is the community attitude towards the concept of software architecture style?* Software architecture is taught in universities. Technical experts and master coaches promote advanced styles. However, what does a typical software engineer think about this concept? We try to answer this question in Section 3 using a developers' survey.

RQ2: *How can we detect a software architecture style in code?* Results of the RQ1 survey encouraged us to try to construct a procedure for detecting software architecture styles in an actual source code. To do so, we first needed to select features related to particular styles using which we can automatically detect them. Section 4 answers the second research question and presents style features and automated scripts which help us to detect styles in code.

RQ3: *What software architecture styles are popular in open-source projects?* Finally, it is of interest to investigate the source code of existing software to decide what styles are popular. Fortunately, much open-source software is available for researchers in the modern world. Thus, we can mine open-source repositories and apply our architecture style detection tool to them.

This procedure is presented and discussed in Section 5.

Section 6 describes some works related to our research project, while Section 7 concludes this paper and proposes the ideas for further work.

3. Software Architectural Styles (RQ1)

Our first questions were as follows. Whether the concept of software architecture style is familiar to developers? Is this concept considered applicable? What particular styles are familiar to developers and are worth considering in the following steps of our research?

3.1 Architecture Styles Survey

To answer these questions, we provided a developers' survey described in this section.

For our research, we have created a survey[2] using Google Forms[3]. This survey consisted of 3 categories of questions.

[1] GitHub web-page: https://github.com/

[2] It can be found at the web-page of our project: https://pais.hse.ru/en/research/projects/softarchstyles

[3] Google Forms: https://docs.google.com/forms

Demographical questions: These are questions about programming experience, job area, preferences in technologies, and a respondent's frameworks.

General questions about software architecture styles: Whether participant had or had not heard and used the concept of architecture styles in their professional life?

Questions about the set of particular architecture styles we selected for our research: We asked whether the participant knew the name of the style and how he or she thought it is possible to identify that certain style in code.

These questions aimed to find out what community of developers thinks and knows about architecture styles usage and architecture styles identification.

3.2 What Styles did We Select?

We have selected the following eleven software architecture styles for this research:

- Model-View-Controller (MVC) architecture;
- Main and sub-programs;
- Machine-learning-based software;
- Event-driven software architecture;
- Reflection-using software;
- Data-centric software architecture;
- Expert system;
- Cloud-service-based software;
- Software with containerization;
- Aspect-oriented software architecture;
- Reactive-based software architecture.

These particular styles were chosen based on software architecture pattern and style catalogs from foundational literature of the field [3-7]. Usually, software architecture books are large and contain profound discussions on each of the styles considered important by book authors. The list of software architecture styles is a massive one. We had to limit this list somehow for it to be treatable within a single research project's borders. To select the particular set of styles, we consulted with literature of the field [3-7] as well as Wikipedia.org information[4]. Some of these styles (for example, MVC and Event-driven architectures) are popular and frequently used among software developers. Others (for example, aspect-based software and expert systems) are not famous in modern software engineering. Besides, we selected styles for which we can define features based on which the style smells can be detected in source code. Thus, we consider it worth investigating this particular set of styles. However, we do not state that this is an exhaustive set.

3.3 Survey Data

The survey was held from September till December 2020. As it has been mentioned, Google Forms were used for the survey. The survey form was spread in different developer communities connected with various areas of development: game development, back-end development, front-end development, data science, etc. We hoped to achieve randomness and broader coverage by doing so. In total, 111 developers participated in the survey.

Participants of the survey have different experiences in software programming. Fig. 1 shows participant programming experience in years. From this figure we can conclude that about half of all participants were in the middle of the experience range: slightly less than one quarter have experience from 1 to 2 years, a little bit more than one-quarter of the total have experience from 3

[4] See page https://en.wikipedia.org/wiki/List_of_software_architecture_styles_and_patterns which itself refers to the paper of Sharma et al. [8].

to 5 years. Experienced developers make one-third of the total number: about one-fifth have experience from 6 to 10, and slightly more than 15% have experience from 11 to 20 years. At the ends of the distribution, we can observe 5% of developers with experience less than 1 year and about 3% of very experienced developers who are in the field for more than 35 years.

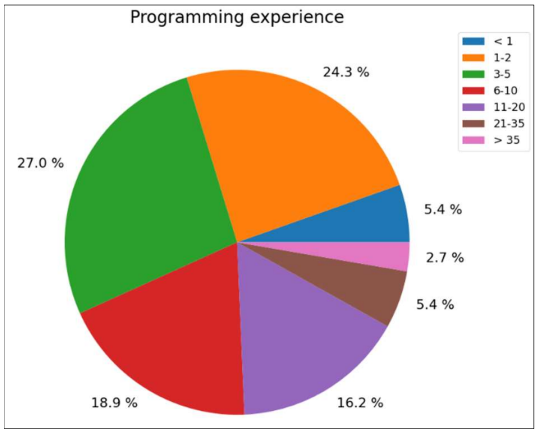

Fig. 1. Participant programming experience in years

Fig. 2 represents fields of software engineering which participants selected as their primary occupation. Note that a participant could select several fields as their primary occupations. We can conclude that survey participants in different areas, with most of them, are back-end developers. The top 7 categories of participant job areas were: back-end development (65.8%), front-end development (34.2%), mobile development (22.5%), data analytics (19.8%), machine learning (18.9%), research (18%), and game development (9.9%).

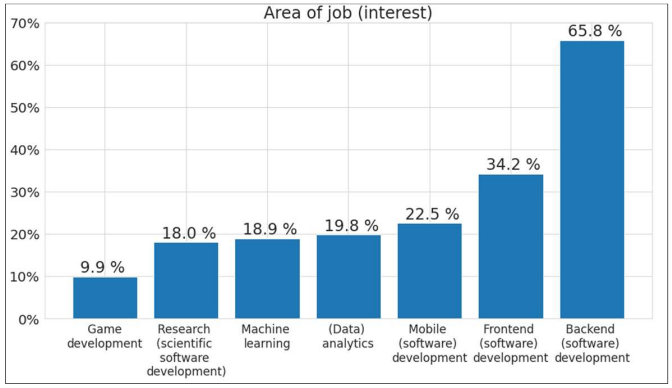

Fig. 2. Participant occupations

Finally, we asked participants about the programming languages they used in their work. Fig. 3 show how they answered. It can be seen that the survey participants are using different languages in their practice. The top three most popular languages in our survey are Python (45.9%), Java (33.3%), and SQL (32.4%). Partially because of these results, we decided to continue our research based on Java and Python source code.

Demographic data showed that our survey participants were similar to the typical software developers. For example, the participants' set of main languages is very similar to the well-known TIOBE Index [5]. Our selection is somehow shifted to object-oriented languages for back-end development. However, of the top 10 languages in TIOBE Mar 2021 (C, Java, Python, C++, C\#, Visual Basic, JavaScript, PHP, Assembly language, SQL) 7 are also presented in the top 10 languages used by survey participants. Developers came from different fields, which are popular in modern software engineering.

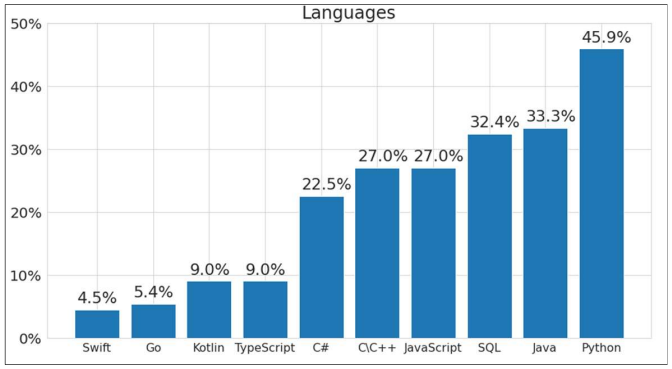

Fig. 3. Participant main programming languages

3.4 Survey Results

Our survey asked whether participants used the concept of software architecture style in their daily work practice. Fig. 4 shows how they answered this question. In this figure, we can see that almost 40\% frequently use the concept of architecture styles in development. 36% of all participants use them from time to time, and one quarter does not use architecture styles at all.

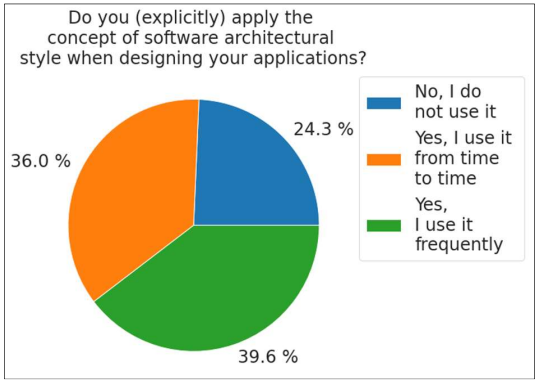

Fig. 4. Do participants apply the concept of software architecture style in their work?

The next question brought us surprising results. We decided to find out what participants thought about the developers' community in general. In particular, we asked what participants thought about how their colleagues applied the concept of software architecture style in their work? Fig. 5 shows that only 14.4% think that developers from their community do not use the concept of architecture

[5] TIOBE Index: https://www.tiobe.com/tiobe-index/

styles. Interestingly, developers tend to think their colleagues are significantly more familiar with the concept of software architecture style than themselves.

Finally, we were interested in what participants think about the feasibility of detecting architecture styles. The developers were asked whether they thought it is possible to identify the usage of a software architecture style in the source code automatically or manually.

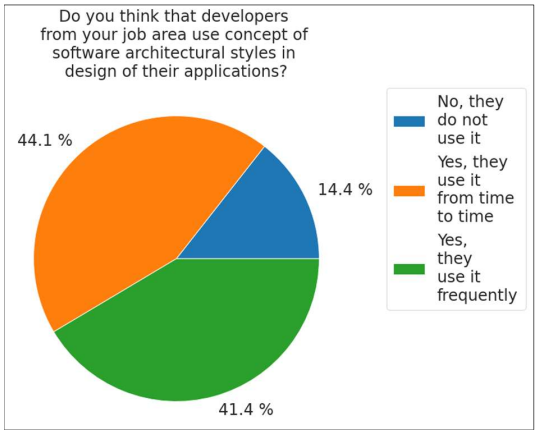

Fig. 5. What participants think about how their colleagues apply the concept of software architecture style in their work?

For each of the styles there were five possible answers as follows:

- Yes, by looking at language constructions manually;
- Yes, by looking at language constructions automatically;
- Yes, by looking at frameworks (you can list frameworks in ``other'' section);
- haven't used this style;
- Other (open answer).

A participant was able to select several answers simultaneously.

Fig. 6 summarizes the answers. To make the figure more illustrative we merged all the answers into the three categories:

- Manually: variant 1) and some of variant 5);
- Auto: variant 2), variant 3), and some of variant 5);
- Have not used: variant 4).

Fig. 6. What participants think about whether particular software architecture styles can be detected, or not?

Numbers in fig. 6 show how many developers selected this particular answer category for a particular architecture style. According to our data, developers are not very optimistic regarding

software architecture style detection. However, for most architectural styles, at least one-third of all survey developers believe they can be identified either automatically (30% – 50%) or manually. Developers tend to think that familiar styles are more likely to be identified. For example, MVC is the most known style among the others. Most developers think it can be identified manually (53.2%) and automatically (51.4%). Expert systems and aspect-oriented software are unfamiliar to more than half of the developers from our selection. Not so many participants believe these styles can be identified by investigating the software source code.

3.5 Conclusions

The results of the developers' community survey are interesting events separately. However, we analyze them in the context of a larger project.

The survey shows that 3 of 4 typical developers apply the concept of software architecture styles in their practice. Moreover, developers believe their colleagues use this concept even more often. This means that developers consider the concept important and valuable in the software engineering process.

From 3.5 to 4 out of 10 typical developers believe that software architecture style can be identified by investigating the software source code. In general, slightly more developers think that a style can be identified manually. Besides, the more familiar a particular style is to the developer, the more likely it would be considered identifiable by this developer.

Thus, it would be interesting to investigate if software architecture styles can be identified using an automated procedure, how this can be done, and what styles are more prevalent in open source.

Note that we can somehow estimate styles' popularity by comparing developers' numbers unfamiliar with different styles. However, we believe that such research based on survey data only would be insufficient.

4. Architecture Style Identification (RQ2)

4.1 Detection Methods and Data Sources

In this research open-source software repositories were used as data. We chose 10 technological communities and companies with extensive lists of open-source repositories on GitHub, which is the largest resource with open software sources. Repositories related to the following companies' Github accounts are considered in this paper:

1) Adobe – https://github.com/adobe,
2) Amazon – https://github.com/amzn,
3) Amazon Web Services – https://github.com/aws,
4) AWS Labs – https://github.com/awslabs,
5) Apache Foundation – https://github.com/apache,
6) Apple – https://github.com/apple,
7) Google – https://github.com/google,
8) IBM – https://github.com/IBM,
9) Microsoft – https://github.com/microsoft,
10) 18F – https://github.com/18F.

We decided to consider only repositories with the code written in Python and Java as these two programming languages are among the most popular according to both well-known indices[6] and to our preliminary developer survey.

[6] For example, see TIOBE Index here: \ https://www.tiobe.com/tiobe-index/

The crawler was written in Python 3. We used the library, called PyGitHub[7]. Every crawler gets access to the companies' repositories by tokens previously generated by us manually on Github.

Our crawlers got access to GitHub repositories by using the token mechanism. Every token allows making ten thousand requests to Github per hour. For the mining process to continue flawlessly, several tokens have been used. The tool iterates through the token list and requests the source code from every repository taken for the research. For every software architecture style, we created a separate specific crawler. Their code is accessible at the project web page.

4.2 Features of Software Architecture Styles

We have created features of different origins for eleven styles from our research. These features can be grouped into two main categories.

The first group of features contains *framework-based* features. We firstly identified frameworks that propose implementations of particular architectural styles. After that, we identified usage of the style by finding usage of these frameworks in source code. Such features were used when identifying Model View Controller (4 python frameworks, 4 java frameworks), Machine Learning based style (24 python frameworks, 11 java frameworks), Event-driven (10 python frameworks, 8 java frameworks), Data-centric (25 python frameworks, 22 java frameworks), Expert systems (7 python frameworks, 3 java frameworks), Cloud systems (10 python frameworks, 7 java frameworks), Aspect-based applications (3 python frameworks, 1 java frameworks).

The second group of features contains *language-based* features. This means that we first identified how certain styles are implemented in specific languages (Java, Python). After that, we identified usage of the style by finding particular language constructs. These features were used when identifying Main and Sub-programs, Reflection architecture styles.

Table 1 provides a short description of every architecture style we have chosen for our research. It is assumed to hint about how they are presented in books and online resources. Besides, we give examples of features that we have used to identify the architectural styles. The complete list of features we used is available on the project web page.

Table 1. Empirical features of eleven software architecture styles

Architecture style	**Short description**	**Examples of empirical features**
Model-View-Controller	Architecture style includes: model (a dynamic data structure), view (a component to represent the information), and controller (this component accepts user input and converts it to commands).	***Python, Spring***: @Controller, import org.springframework.stereotype.Controller ***Java, Django***: from django.db import models, from models import
Main and sub-programs	Architecture style assumes an absence of classes. It means that application use only functions/ procedures and may use classes only as storage for functions/procedures without creating instances of classes.	***Python features:*** def main, fwithout defg main() ***Python anti-feature:*** def __init__ ***Java feature:*** public static void main(String[] args), ***Java anti-features:*** class fClassNameg, new fClassName)
Machine-learning based	Architecture style assumes usage of any data science-related frameworks and libraries.	***Python, Scikit learn:*** from sklearn, import sklearn ***Java, Apache Spark ML-lib:*** org.apache.spark.mllib

[7] PyGitHub web page: {https://pygithub.readthedocs.io

Event-driven software	Architecture style implies production, detection, consumption and reaction to events. Usually, implemented based on special frameworks	***Python, Apache Kafka:*** from kafka, import kafka ***Java, Apache Kafka:*** org.apache.kafka
Reflection-using software	Architecture style assumes that application's processes can and do examine, introspect and modify their own structure and behavior	***Python features:*** type(obj), isinstance(obj, obj) ***Java features:*** java.lang.reflect, .getClass()
Data-centric software	Architecture style implies that database is a crucial (central) part of application	***Python, MySQL:*** from mysql, import mysql ***Java, MySQL:*** import java.sql, com.mysql.jdbc.Driver
Expert system	Architecture style assumes usage of any expert system frameworks and libraries as a part of the considered software.	*Python, Experta:* from experta, import experta ***Java, Apache Jena:*** import org.apache.jena
Cloud-service-based	Architecture style implies usage of frameworks and libraries, which let usage of cloud based delivery and inter-cloud network.	***Python, Apache Libcloud:*** from libcloud, import libcloud ***Java, Google Cloud:*** com.google.cloud
Software with containerization	Architecture style assumes usage of frameworks and libraries which let usage of virtual machines.	***Python, VMWare:*** from vmware, import vmware ***Java, VMWare:*** com.vmware
Aspect-oriented software	Architecture style aims to increase modularity by allowing separation of cross-cutting concerns.	***Python, AspectLib***: import aspectlib, from aspectlib ***Java, AspectJ:*** @Aspect, import org.aspectj
Reactive-based software	Architecture style pays attention to data streams and propagation of change.	***Python, ReactiveX:*** from rx, import rx ***Java, ReactiveX***: import io.reactivex

Investigating Popularity of Particular Styles in Open-source Software (RQ3)}

5. Investigating Popularity of Particular Styles in Open-source Software (RQ3)

5.1 Dataset Description

Our web crawlers gathered a dataset that we used to answer RQ3. This dataset consists of JSON files. Each file in the dataset is related to a triple: (`programming\_language; company\_name; software\_architecture\_style`). In total, the 3057 repositories were processed. 1682 of them are repositories with source code in Java, whereas 1375 contain Python source code. Each repository can contain code in other programming languages as well. The results of mining contain 172 JSON files. These files contain data on features identified for a particular triple. Each file includes a set of pairs (`repository : [found_features]`), where `[found_features]` is a list of features of the particular architecture style which were found in the `repository`.

Here is the example of such a pair: … `"EmbeddedSocial-Android-SDK": ["NONE", "getName\_feature", "getClass\_feature"], …`

Every string includes a constant indicating if the related repository's processing was finished. It was used for repository processing and did not have any special meaning. Some of the lines may contain constant indicating that the mining process was stopped. This happened when a repository weighted too much to be processed by 10 000 requests of the crawler. In these cases, a GitHub API token reaches its' limit. This case was not frequent. In particular, 2162 pairs out of a total 27107 contain these stops.

The full dataset is available on the project web page.

5.2 Data Analysis and Discussion

Summarized results derived from the dataset are presented in this section. The following tables show these results. Let us consider and discuss the popularity of particular styles. Note that open repositories of Apple company contain no source code in Java.

5.2.1 Model-View-Controller (MVC) architecture

In Table 2 one can see that MVC is used in approximately 25% of Microsoft, IBM, and Apache Java repositories. In Java repositories, the MVC style is mostly represented by the usage of the Spring framework that is very popular, especially in Apache Foundation projects.

Table 2. MVC style usage frequency (Java repositories)

	Microsoft	IBM	Google	Awslabs	Aws	Apache	Amzn	Adobe	18F
Processed	118	135	205	75	28	1044	18	53	6
MVC	29	28	4	7	6	274	0	4	0
Spring	29	27	4	1	4	237	0	4	0
Free Marker	0	1	0	0	4	48	0	0	0
Apache Struts	0	0	1	1	0	29	0	0	0

MVC is used in slightly less than 10% of Microsoft, IBM, Google, AWSlabs Python repositories (see Table 3). In Python repositories, MVC style is mostly represented by Django framework. Thus, web development is responsible for a significant fraction of usage cases in Python community. It is also interesting that Python is relatively more popular in open projects of commercial companies, whereas Apache Foundation is the leader in the development of Java projects.

Table 2. MVC style usage frequency (Pithon repositories)

	Microsoft	IBM	Google	Awslabs	Aws	Apple	Apache	Amzn	Adobe	18F
Processed	295	279	337	159	51	20	68	15	26	127
MVC	29	28	4	7	6		274	0	4	0
Django	19	10	20	4	2	1	5	1	1	42
Giotto	7	4	12	9	6	1	4	1	2	8
CherryPy	2	1	2	1	0	0	2	0	0	1
Turbo Gears	0	0	0	0	1	0	3	0	1	0

This style is the second most popular} from all styles in our style set. It is a significantly more popular style than others. This conclusion agrees with the survey results shown in fig. 6.

5.2.2 Main and sub-programs

Let us consider fig. 7 and 8. Each of these two figures shows two intersecting disks. The left one shows the number of repositories with «main» function. The right one shows a number of

repositories without the usage of constructors. Thus, repositories that satisfy our criteria lie in the intersection.

It is easy to see no more than 1 repository of such type in Java. It is not unexpected because Java is a pure object-oriented language. So, any Java program contains objects or classes.

On the other hand, there are about 7% of all Python repositories (59) in which this procedural style was applied.

In general, we can conclude that *this style is not very* popular among open-source repositories from our dataset.

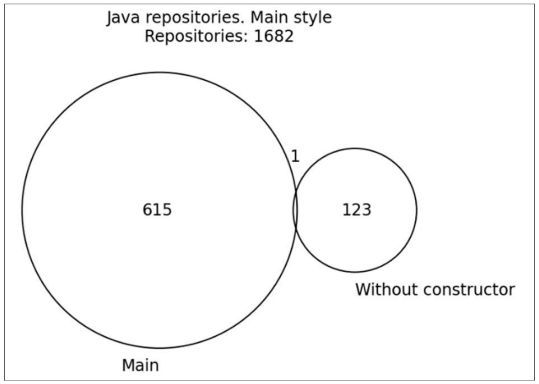

Fig. 7. Main and sub-programs style (Java repositories)

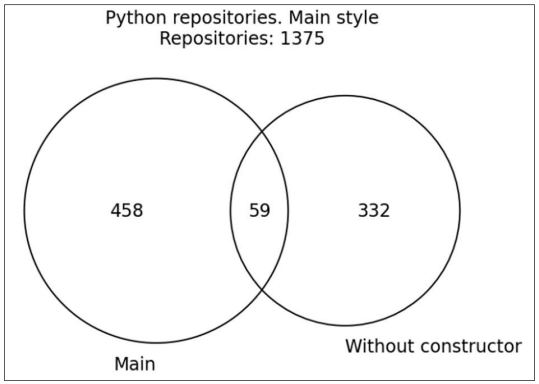

Fig. 7. Main and sub-programs style (Python repositories)

5.2.3 Event-driven software architecture

According to Tables 4 and 5 Event-driven architecture style is used in approximately 9% of IBM and Apache Java repositories. Curiously, event-driven style is applied in more than 50% of AWS Python repositories and approximately 20% of Apache Python repositories. Such applications are related to distributed and asynchronous software for web applications. Other companies tend to apply event-driven style in less than 1% of their Java and Python repositories. Event-driven architectures are mostly represented by the usage of Kafka framework and Amazon Active MQ framework in both Java and Python repositories.

Table 4. Event-driven style usage frequency (Java repositories)

	Microsoft	IBM	Google	Awslabs	Aws	Apache	Amzn	Adobe	18F

Processed	118	135	205	75	28	1044	18	53	6
Event-driven	3	10	0	0	0	88	0	2	0
Kafka	3	9	0	0	0	41	0	2	0
Apache Qpid	1	0	0	0	0	10	0	0	0
RabbitMQ	0	1	0	0	0	6	0	0	0
Amazon ActiveMQ	0	0	0	0	0	41	0	0	0
Apache RocketMQ	0	0	0	0	0	7	0	0	0
Zero MQ	0	0	0	0	0	2	0	0	0

Table 5. Event-driven style usage frequency (Python repositories)

	Microsoft	**IBM**	**Google**	**Awslabs**	**Aws**	**Apple**	**Apache**	**Amzn**	**Adobe**	**18F**
Processed	295	279	337	159	51	20	68	15	26	127
Event-driven	9	11	5	0	32	9	13	1	3	6
Kafka	0	5	0	0	0	0	3	0	0	0
Apache Qpid	0	0	0	0	0	0	3	0	0	0
RabbitMQ	0	1	0	0	0	0	3	0	0	0
Amazon ActiveMQ	8	5	5	0	32	0	6	1	3	6
Apache RocketMQ	0	0	0	0	0	0	1	0	0	0
Zero MQ	1	0	0	0	0	0	1	0	0	0

We can conclude that usage of *event-driven architectures hugely variates* from company to company and *relatively popular in projects* of AWS and Apache whose business is mostly *web-based and large-scale oriented*. Thus, these companies invest in scalable web applications and infrastructure code. Other companies concentrate more on desktop, mobile, and web applications without such need in scaling and asynchronous code.

5.2.4 Machine-learning-based software

The first general finding is that Java is not used commonly to develop machine-learning-based software. According to Table 6 we found smells of machine-learning-based style only in 16 Java repositories. On the other hand (see Table 7), this style is often used in Python repositories by various companies: Microsoft (61%), IBM (52%), Google (38%). ML source code is mostly represented by the usage of Numpy, Pandas, Matplotlib, and libraries for neural networks.

Table 6. Other architecture styles usage frequency (Java repositories)

	Microsoft	**IBM**	**Google**	**Awslabs**	**Aws**	**Apache**	**Amzn**	**Adobe**	**18F**
Processed	118	135	205	75	28	1044	18	53	6
ML-based	1	1	0	3	0	11	0	0	0
Data-centric	21	20	34	15	9	228	1	3	1
Cloud-based	0	0	0	0	25	0	0	0	0
Container	0	0	0	0	0	0	0	0	0
Aspect-oriented	0	0	0	0	0	0	0	0	0

Reactive-based	0	0	0	0	1	0	0	0	0
Expert system	0	0	0	0	1	13	0	0	0

We can conclude that *machine-learning applications are very popular in Python ecosystem*. Most Python repositories of companies contain smells of ML. Moreover, we can conclude that *machine-learning software is the most popular* software style (with respect to a total number of repositories with this style) according to our data.

Table 7. Other architecture styles usage frequency (Python repositories)

	Microsoft	**IBM**	**Google**	**Awslabs**	**Aws**	**Apple**	**Apache**	**Amzn**	**Adobe**	**18F**
Processed	295	279	337	159	51	20	68	15	26	127
ML-based	180	146	129	45	24	13	13	8	12	25
Data-centric	47	25	29	10	3	2	22	0	2	24
Cloud-based	3	1	42	1	0	0	2	0	1	2
Container	1	0	0	0	0	1	1	0	0	0
Aspect-oriented	0	0	0	0	0	0	0	0	0	0
Reactive-based	0	0	0	0	0	0	0	0	0	0
Expert system	0	0	0	0	0	0	0	0	0	0

5.2.5 Data-centric software architecture

We found smells of data-centric style in 15%–25% of Java repositories (Microsoft, IBM, Google, Apache, see Table 6) and in 8%–20% of Python repositories (Microsoft, IBM, Google, 18F, see Table 7). In Java repositories, data-centric software style is mostly represented by PostgreSQL and MySQL libraries' usage. This style is represented by the usage of the SQLAlchemy library in Python repositories.

We can conclude that *this style is the third most popular* of all styles thanks to Apache Foundation with more than two hundred such projects. Other companies apply the style as well.

5.2.6 Reflection-using software

This style was detected using several language features that indicate reflection appliances in a source code. Many repositories contain at least one of the features of a reflective code. However, we believe that the code with such an ephemeral smell can be called reflection-using. However, what should be the number of reflective features in code to call it reflection-using software. It is not that easy to define the concrete number. Thus, we decided to show the summarized data in this paper. A better definition of this architecture style will be a subject for future work.

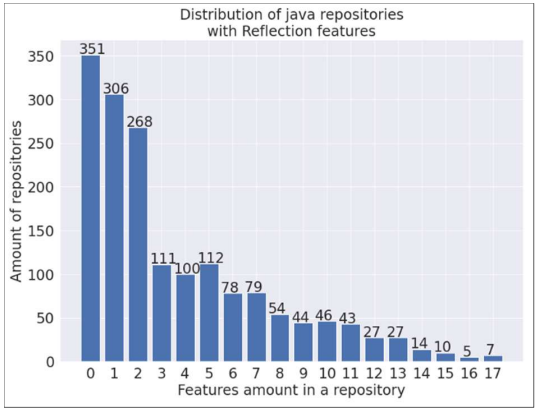

Fig. 9. Reflective code features in Java repositories

Fig. 9 shows the results for Java repositories, whereas fig. 10 considers Python repositories. In both cases, one can see that about 20% of repositories contain no reflective code smells. So, we can conclude that about 80% of Java and Python repositories have at least one feature of Reflection-using software.

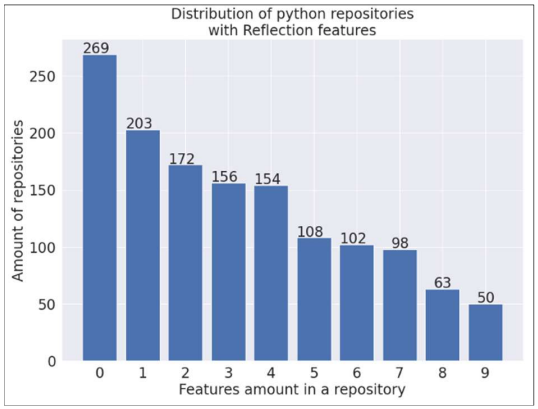

Fig. 9. Reflective code features in Python repositories

Our conclusion is that *most of the open-source software in our dataset contains some reflective code features*. Our definition for this style is too vague and has to be refined.

5.2.7 All other styles: cloud-service-based, aspect-oriented, reactive-based software, expert systems, software with containerization

It is clearly seen in tables 6 and 7 that smells of all other software architecture styles are very uncommon in our dataset.

Cloud-service-based style tends to appear in AWS Java repositories. This can be explained by the usage of AWS's own library for cloud development. Also, the Cloud-service-based style was found in Google Python repositories. It can be explained by the usage of Google's library for cloud development. These two cases are outliers, and overall we did not find out Cloud-service-based style as a popular one.

The same is true for other styles. We did not find almost any usage of these styles with the proposed features. Thus, we can conclude *all these software architecture styles are unpopular* in open-source repositories of our dataset.

This can be due to at least two reasons: either these styles are uncommon in open-source software, or we use flawed features. Both reasons are possible. The future work will be to elaborate on this issue.

5.3 Additional Results Validation

To verify the proposed feature model, we decided to ask developers of repositories, which we have processed, about the usage of the 11 architecture styles in their repositories. We extracted developers' emails from each repository we have processed. There were about 10 thousand repositories. Then we used Python code to send every one of them a letter with a link to the particular survey based on Google Forms. This form asked developers to specify what of our 11 software architecture styles they used in their repository. We got 69 replies to our Google Form. Out of these replies we extracted information on 78 repositories that we have previously processed.

One can easily see that we have no significant number of answers here. Thus, the following can not be considered as an extensive validation. However, we believe these results still can be of interest to the reader.

On every architecture style out of 11 we counted 4 metrics: accuracy, precision, recall, and F1-score. We considered developers' answers from the form as correct data and our answers as predictions. Among the repositories that authors answered our survey, there was no that used the following styles: Main and sub-programs, Expert system, or software with containerization.

Table 8 shows the results. According to the table, the best F1-score was reached for Reflection-using software (0.58) and Data-centric software (0.45). Recall overall was less than 30\% with such exceptions as Reflection-using software (0.64), Model-View-Controller (0.39), and Data-centric software (0.39). The highest precision was achieved for Event-driven software (0.75).

Table 8. Additional results validation

	MVC	**Main**	**ML-based**	**Event-driv.**	**Reflect.**	**DB-centr.**
Accuracy	0.65	0.92	0.79	0.51	0.58	0.72
Precision	0.41	—	0.44	0.75	0.53	0.53
Recall	0.39	0	0.27	0.14	0.64	0.39
F1	0.4	—	0.33	0.24	0.58	0.45
	Expert Sys.	**Cloud-based**	**Container**	**Aspect**	**React.-based**	
Accuracy	0.88	0.56	0.62	0.76	0.68	
Precision	—	0.5	—	1	0.67	
Recall	0	0.09	0	0.14	0.08	
F1	—	0.15	—	0.25	0.14	

The results are different for various architectural styles. We can conclude the following.

Features for Main and sub-programs style and Expert systems could not be validated because among the repositories from the validation survey, there was no use of these two styles. Features for software with containerization style are not full. Using our features, we did not find it in any of the repositories in which the style was used according to their developers. Features for Reflection-using software and Data-centric software styles have not been enough for perfect identification, but they showed appropriate F1-score results. Features for Model-View-Controller, Machine-learning-based, Event-driven, Cloud-service-based, Aspect-based, and Reactive-based software styles show bad

performance, mostly because of low recall. This means that our features have not fully covered the usage of these styles, and further investigation is needed.

5.4 Conclusions

Most of the results obtained by our automatic crawlers agree with the survey results, which are shown in fig. 6. Less-known styles are less common in open-source repositories; well-known styles can be found in many more repositories.

An outlier here is *software with containerization* style. Feature for this style seems ill-designed because many people are accounted for it, whereas we can not detect it in source code.

Both an automated analysis and a survey indicate aspect-based software and expert systems as the *least popular* architecture styles.

However, the additional validating survey (with a small number of answers) indicated that our features for some architecture styles show lousy performance. Thus, additional work is needed to improve the style and feature sets.

6. Related Work

We consider two large fields as related to our research. These fields are *software architecture research* and *software repository mining*. Whereas the former field is relatively old in terms of software engineering time scale, the latter is relatively young and fast-growing. We will try to observe both fields in this section.

Sharma, Kumar, and Agarwal [8] listed 23 software architecture styles in 6 categories due to the application type. This paper can be considered as a starting point to discuss architecture styles. The authors have chosen some styles (what styles?) out of all mentioned and gave short descriptions to them. However, there cannot be observed any code features of any style which can be used to identify it in a real project. The paper also leaves without attention statistical aspects of architecture style popularity in practice.

Automated software architecture recovery is related as well. Researchers in this field aim at constructing models of architecture decisions of existing software using data analysis and other automated techniques [9].

In software repository mining papers on code smell detection are close related to our project. Fontana et al. [10] concentrated on code smells and a machine learning-based approach to code smells detection. The authors collected a dataset of heterogeneous systems and a set of tools for detecting code smells and trained different machine learning algorithms with default parameters. Boussaa et al. [11] introduced code smell detection based on genetic algorithms that are called the competitive-co-evolution-based method. The method's idea is to generate two data samples: a sample of code smells and a sample of solutions. The aim of code smells generation is to escape from search methods, and the solution aims to cover more code smells. These works do not pay attention to software architecture styles, but their general approach seems attractive.

A repository mining method has been applied to reveal how software architecture evolves with time [12]. Code mining can help to evaluate software architecture as well [13]. Kouroshfar et al. \[14] applied automated architecture recovery techniques to show how the erosion of software architecture decisions influences software evolution.

There is a massive corpus of literature on software architectural smells and their automated detection. Architectural smells are signs of bad practices in the software design process, similar to code smells. The difference is that architectural smells are related to the level of general design decisions, whereas code smells are related to anti-patterns and bad practices on the level of software code. Fontana et al. [15] investigated how these two types of smell are interrelated. Previously, many automated tools have been developed to detect or predict architectural smells [16-20]. Azadi et al. [21] even proposed a catalog of such smells which different tools can detect. Features of software

architecture styles that we consider in this paper are similar to smells. However, our features do not sign *bad* practices or anti-patterns. Contrariwise, our features indicate the presence of an architectural style.

Note that surveys are considered a good research tool in empirical software engineering. For example, Palomba et al. [22] used surveying to understand how developers feel a relationship between code and community smells. In our research, we apply surveys as well.

Recently, repository mining has been used to explore software in an empirical study on what software project artifacts are [23]. Not surprisingly, software projects consist of code but also of documentation, data, and many more different artifacts. Our research is similar in the sense of intentions. We seek for better understanding of the current field of software development.

6. Conclusions and Further work

In this paper, we measured industrial software developers' attitudes to the concept of software architecture style. We also investigated the popularity of eleven concrete architecture styles.

We found that the notion of software architecture style is not just a concept of academics at universities. Programmers apply this concept in their work. Moreover, industrial software developers consider the concept as improving their professional skill-set.

We formulated features for eleven concrete software architecture styles and developed crawlers based on these features. The results of repository mining using the features show that the most popular styles among developers of open-source projects are machine-learning-based software, Model-View-Controller architecture, and Data-centric software architecture.

We additionally validated the results obtained by crawlers using a special developer survey.

This validation shows that features for some architecture styles are ill-defined and have to be improved.

This paper presents up-to-date results of our research project. We plan to continue the project to understand the concept of software architecture style better. Updates can be found at the project web page: https://pais.hse.ru/en/research/projects/softarchstyles.

The set of software architecture styles we used in the paper is not comprehensive. It is possible to modify and extend it based on this work's results. This will be one of the directions of our future work.

Besides, the dataset gathered by our crawlers is related to a limited set of open-source repositories related to large software communities and companies. It is possible that our results are somehow biased and overfitted to this particular dataset. So, additional research is needed based on wider datasets.

Particular software architecture styles are still not sufficiently well-defined. Some of them – like reflection-using software --- need better and clearer definitions to deal with them in a less vague manner. We believe it is possible to construct concise and rigorous definitions based on more profound empirical research results.

References

[1] P. C. Clements and M. Shaw. “The Golden Age of Software Architecture” revisited. IEEE Software, vol. 26, no. 4, 2009, pp. 70-72.

[2] M. Shaw and D. Garlan. Software architecture - perspectives on an emerging discipline. Prentice Hall, 1996, 264 p.

[3] R. N. Taylor, N. Medvidovic, and E. M. Dashofy. Software Architecture - Foundations, Theory, and Practice. Wiley, 2010, 750 p.

[4] M. Richards and N. Ford. Fundamentals of Software Architecture: An Engineering Approach. O’Reilly, 2020, 432 p.

[5] M. Richards. Software architecture patterns. O’Reilly Media, 2015, 47 p.

[6] M. Kleppmann. Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems. O'Reilly Media, 2017, 616 p.
[7] L. Atchison. Architecting for Scale: High Availability for Your Growing Applications. O'Reilly Media, 2016, 230 p.
[8] A. Sharma, M. Kumar, and S. Agarwal. A complete survey on software architectural styles and patterns. Procedia Computer Science, vol. 70, 2015, pp. 16-28.
[9] A. Shahbazian, Y. K. Lee et al. Recovering Architectural Design Decisions. In Proc. of the 2018 IEEE International Conference on Software Architecture (ICSA), 2018, pp. 95-104.
[10] F. A. Fontana, M. Zanoni et al. Code smell detection: Towards a machine learning-based approach. In Proc. of the 2013 IEEE International Conference on Software Maintenance, 2013, pp. 396-399.
[11] M. Boussaa, W. Kessentini et al. Competitive coevolutionary code-smells detection. Lecture Notes in Computer Science, vol. 8084, 2013, pp. 50–65.
[12] D. M. Le, P. Behnamghader et al. An empirical study of architectural change in open source software systems. In Proc. of the 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, 2015, pp. 235–245.
[13] L. Zhu, M. A. Babar, and D. R. Jeffery. Mining patterns to support software architecture evaluation. in WICSA. In Proc. of the Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA 2004), 2004, pp. 25-36.
[14] E. Kouroshfar, M. Mirakhorliet al. A study on the role of software architecture in the evolution and quality of software. In Proc. of the 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, 2015, pp. 246-257.
[15] F.A. Fontana, V. Lenarduzzi et al. Are architectural smells independent from code smells? An empirical study. Journal of Systems and Software, vol. 154, 2019, pp. 139-156.
[16] F.A. Fontana, I. Pigazzini et al. Automatic detection of instability architectural smells. In Proc. of the 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2016, pp. 433–437.
[17] F.A. Fontana, I. Pigazzini et al. Arcan: A tool for architectural smells detection. In Proc. of the 2017 IEEE International Conference on Software Architecture Workshops (ICSAW), 2017, pp. 282–285.
[18] A. Biaggi, F. A. Fontana, and R. Roveda. An architectural smells detection tool for C and C++ projects. In Proc. of the 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2018, pp. 417–420.
[19] U. Azadi, F. A. Fontana, and M. Zanoni. Machine learning based code smell detection through WekaNose. In Proc. of the 40th International Conference on Software Engineering: Companion Proceeedings, 2018, pp. 288–289.
[20] F. A. Fontana, P. Avgeriou et al. A study on architectural smells prediction. In Proc. of the 2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2019, pp. 333–337.
[21] U. Azadi, F. A. Fontana, and D. Taibi. Architectural smells detected by tools: a catalogue proposal. In Proc. of the 2019 IEEE/ACM International Conference on Technical Debt (TechDebt), 2019, pp. 88–97.
[22] F. Palomba, D. A. Tamburri et al. How do community smells influence code smells? In Proc. of the 40th International Conference on Software Engineering: Companion Proceeedings, 2018, pp. 240–241.
[23] R. Pfeiffer. What constitutes software? An empirical, descriptive study of artifacts. In Proc. of the 17th International Conference on Mining Software Repositories, 2020, pp. 481–491.

Информация об авторах / Information about authors

Алексей Александрович МИЦЮК, кандидат компьютерных наук, доцент, старший научный сотрудник. Научные интересы: извлечение и анализ процессов, информационные системы, архитектура программного обеспечения, сети Петри

Alexey Alexandrovich MITSYUK, PhD in Computer Science, Associate Professor, Senior Research Fellow. Research interests: process mining, information systems, software architecture, Petri nets.

Николай Арсенович ЖАМГАРЯН, бакалавр программной инженерии, НИУ ВШЭ, студент магистратуры, университет Мичигана, ИТ аудитор, КПМГ СНГ. Научные интересы: наука о данных, обработка естественного языка, машинное зрение, машинное обучение, глубокое обучение.

Nikolay Arsenovich JAMGARYAN, Bachelor of Software Engineering, HSE, Master's student, University of Michigan, IT auditor, KPMG CIS. Research interests: data science, natural language processing, computer vision, machine learning, deep learning.