software

MDPI

Article

Approach to Formalizing Software Projects for Solving Design Automation and Project Management Tasks

Aleksey Filippov *, Anton Romanov, Anton Skalkin, Julia Stroeva and Nadezhda Yarushkina

Department of Information Systems, Ulyanovsk State Technical University, 32 Severny Venetz Street, 432027 Ulyanovsk, Russia

* Correspondence: al.filippov@ulstu.ru; Tel.: +7-908-485-8390

Abstract: GitHub and GitLab contain many project repositories. Each repository contains many design artifacts and specific project management features. Developers can automate the processes of design and project management with the approach proposed in this paper. We described the knowledge base model and diagnostic analytics method for the solving of design automation and project management tasks. This paper also presents examples of use cases for applying the proposed approach.

Keywords: design automation; project management; software system; time series; knowledge base; software repository

Citation: Filippov, A.; Romanov, A.; Skalkin, A.; Stroeva J.; Yarushkina, N. Approach to Formalizing Software Projects for Solving Design Automation and Project Management Tasks. *Software* **2023**, *2*, 133–162. https://doi.org/10.3390/software2010006

Academic Editor: Sanjay Misra, Robertas Damaševičius, Bharti Suri

Received: 30 December 2022
Revised: 3 March 2023
Accepted: 6 March 2023
Published: 8 March 2023

1. Introduction

The development of a modern software system is impossible without the construction and practical use of its architectural description (AD). The AD is in demand at all stages of a software system life cycle. The AD is the first (earliest) representation of a software system. The AD can be verified (tested) as a complete system. Moreover, the most significant requirements and restrictions are stated in the AD, ensuring that everyone considers and understands the concerns of stakeholders in the project.

Developers must comply with the requirements in the AD without fail in all following versions of the software system. Compliance ensures the integrity of the project. Developers can change the AD in the software system development, but only if there are very solid reasons.

Thus, the AD captures the high-level requirements and their corresponding decisions, which may not be changed at lower levels of the project, because changes to the AD are too costly.

The responsibility for the quality of the developed software system lies with the architects and the project managers. The specifics of the creation of modern software systems include the intensive use of software engineering in a highly complex computerized operating environment. The architects and developers consistently apply a heterogeneous experience when interacting with that environment. Development practice shows that these conditions contribute to the negative manifestations of the human factor, among which costly faults and design errors are especially undesirable.

According to the results of research by the Standish Group Corporation, regularly published since 1994, the success rate of projects has slightly more than doubled from 16% in 1994 to the present [1].

Developers are involved in three key processes when developing software [2,3]:

1. Understanding the context of some problem area (domain).
2. Designing a domain model and design space.
3. The formation of some understanding of the context as design artifacts.

Designers need to highlight the entities and business processes from the domain in the first process. Those entities and business processes are important for solving development tasks and determine the significant properties of these objects. Designers form the operational space (OS) of design activity because of their understanding of the domain.

The designers form the conceptual space (CS) of design activity in the second process. Developers form the CS in the design process because of their mental activity based on their experience and understanding of the OS.

The developers form the materialization of the contents of the CS as a set of design artifacts because of the third process.

Currently, there is a large amount of research in software engineering. However, most of the activities of software developers in designing and constructing software are based on the experience gained from working on previous projects. Various functional and non-functional requirements for the software project often affect the development results. Thus, the formation of a coherent theoretical base to support the design and construction of any software is a complicated task [2,3]. At the moment, developers are using a basic theory of software engineering and various theories focused on the development of software in various classes.

Moreover, developers handle resource limitations when developing any software [4]. The project manager needs to evaluate the status of the project to make timely management decisions. The quality of management decisions directly affects the quality of the design artifacts and the quality of the SS.

In most cases, planning problems arise because of the following circumstances [4]:

- The designers did not fully form the CS at the initial stages of the project;
- The designers did not discuss the functional requirements with the customer;
- There is not enough time to conduct usability testing.

Thus, we can define the following objectives of this study:

- It is necessary to develop a model and methods for building a knowledge base to collect the experience of previous projects to support the processes of software design and construction;
- It is necessary to develop a method of diagnostic analytics for the evaluation of the project development processes to improve the quality and efficiency of management decisions.

Thus, the main problems of the modern development of software systems are as follows [5–8]:

1. A high level of uncertainty when a project is developed for a new domain or when using new architectural approaches or technologies.
2. The influence of the external environment on the development process, including an unexpected reduction in resources.
3. A lack of necessary competencies among team members.
4. The need for the rapid assessment of numerous factors affecting the success of the project and the quality of project management decisions.

In this paper, we consider the experience of previous projects as a set of design artifacts. We understand a set of quantitative indicators from the task tracker as the key features of the project development processes, for example, the number of error notifications, the average time to close an error notification, the team size, etc.

We present this paper as the following sections. Section 2 contains the review of the works in this study for a better understanding of the problems and objectives. Section 3 presents a description of the proposed knowledge base model to consider the experience of previous projects and a description of the diagnostic analytics method to support the software development process. Section 4 presents examples of use cases for applying the proposed approach. Section 5 contains discussion. This paper ends with the conclusions.

2. State Of The Art

Different researchers studied software design automation in various works. In most cases, they propose models and methods for representing experience and knowledge to organize corporate knowledge bases by formalizing design artifacts of various types [3,9–14].

The paper [3] considered the question–answer protocol for the case-based support of the design process. The author focused on the description of methods for solving various problems of designing and constructing software with a question-answering method (WIQA). The WIQA is a complex of methods and means that create and use QA-models for project tasks solved at the conceptual stage of software system designing. The primary applications of the WIQA are the iterative creation of QA-nets, the control distribution of the tasks of the tree in its current state among members of the team, and solving the tasks using stepwise refinement based on the question–answer analysis.

The authors of the paper [9] considered the software project as a set of different contexts that compose the description of the software architecture under the ISO/IEC/IEEE 42010:2022 standard [15]. The authors also presented in this paper the following metamodels:

- A metamodel for the scope model kind;
- A metamodel for the user model kind;
- A metamodel for the environment model kind.

The authors of the paper [10] considered a software project as a fragment of an ontology specific to a domain of this project. That ontology contains a set of the key concepts related to a software system domain, a set of key concepts extracted from a source code to establish a semantic relationship between different concepts. In addition, the proposed ontology contains common knowledge from the General Software Engineering domain as a basis.

In papers [11,12], the authors described an algorithm that forms an ontology of a software project based on the analysis of a set of UML diagrams to identify various design patterns. The authors use design patterns that extract during the analysis for searching projects with structure similarity, considering their linkage to a specific domain.

In paper [13], the authors proposed a mathematical apparatus for representing the CS using fuzzy logic methods.

The authors of the paper [14] described the fuzzy ontology to structure the knowledge associated with non-functional requirements via a fuzzy ontology. The approach is based on the use of the fuzzy ontologies for modeling knowledge that relates non-functional requirements to design patterns and to the families they belong to. That approach allows for the representation and maintenance of the knowledge by keeping the flexibility and fuzziness of modeling.

We also considered papers about the analysis of open-source software repositories to evaluate the quality of the repository, depending on various design, construction, and project management practices [16–21].

The authors of the paper [16] studied the impact of using a test-driven development methodology on various project quality indicators: the number of test files, average commit velocity, number of bug-referencing commits, number of issues recorded, usage of continuous integration, number of pull requests, and distribution of commits per author.

The authors of the papers [17,18] analyzed the configuration files for the Docker environment (Dockerfiles) to find successful design solutions (best practices) in them.

The authors of the paper [17] introduced a novel rule-mining technique. Through this automated mining, they could extract 16 new rules that were not found during manual rule collection. In addition, the authors manually collected a set of rules for Dockerfiles from commits to the files in the Gold Set.

The authors of the paper [18] parsed Dockerfiles specified in a declarative language and enriched them with information about changes. In addition, they captured the information of files that were changed in a commit near a Dockerfile change.

In the paper [19], the authors selected project indicators based on the developer survey and proposed a forecasting method based on a combination of machine learning methods.

The proposed method allows for predicting the quality of a project in the future (health indicators prediction).

In papers [20,21], the authors empirically evaluated the impact of different community organization and project management styles on project quality.

The authors of the paper [20] investigated the relation between community patterns and smells, with the purpose of understanding whether the structural organization of a community might lead to some sort of social debt.

The authors of the paper [21] proposed YOSHI (Yielding Open-Source Health Information), a tool able to map open-source communities onto community patterns, sets of known organizational and social structure types, and characteristics with measurable core attributes.

In our opinion, it is necessary to form a knowledge base when solving the problem of using the experience of previous projects for design automation and project management. The knowledge base should consider various aspects of the software project and the features of the development process. Most importantly, all this information must be collected and analyzed, considering the dynamics of the development of the project during its life cycle. The following works influenced our study:

- Ralph P. The sensemaking-coevolution-implementation theory of software design [2].
- Sosnin P. Substantially evolutionary theorizing in designing software-intensive systems [3].
- Bedjeti A.; Lago P.; Lewis G.A.; De Boer R.D.; Hilliard R. Modeling context with an architecture viewpoint [9].
- Di Noia T.; Mongiello M.; Nocera F.; Straccia U. A fuzzy ontology-based approach for tool-supported decision making in architectural design [14].
- Schermann G.; Zumberi S.; Cito J. Structured information on state and evolution of dockerfiles on GitHub [18].
- Xia T.; Fu W.; Shu R.; Agrawal R.; Menzies T. Predicting health indicators for open source projects (using hyperparameter optimization) [19].

Thus, it is necessary to take a comprehensive approach to data collection. We need to consider the following:

- Design artifacts;
- The project's compliance with the requirements and constraints of some domain;
- The influence of various indicators and management decisions to the project development process;
- Their cumulative influence on each other.

3. Materials and Methods

This section discusses the proposed models, methods, and algorithms for automating the design and management of software projects.

Modern software development practices are mainly based on iterative (flexible) development methodologies that allow the following [3,4]:

1. The ability to quickly respond to changing customer requirements;
2. An operative demonstration of the new software functionality to customers for evaluation, clarifications, and adjustments;
3. An increase in the efficiency of managerial decisions.

Moreover, developers use the Design Thinking methodology (DT) in the software development process [2–4]. The key feature of DT is the solution of engineering, business, and other problems, based on a creative, rather than analytical, approach. When using DT, developers do not solve problems based on critical analysis, but consider them as a creative process, which allows them to find unexpected and non-obvious solutions.

The DT methodology contains the following stages of solving the problem [2–4]:

1. Problem definition.
2. Researching.

3. The formation of ideas.
4. Prototyping.
5. Choosing the best solution.
6. Implementation.
7. Evaluation.

In this article, we consider each iteration of the flexible development process as the following steps (Figure 1):

1. Planning.
2. Design.
3. Construction.

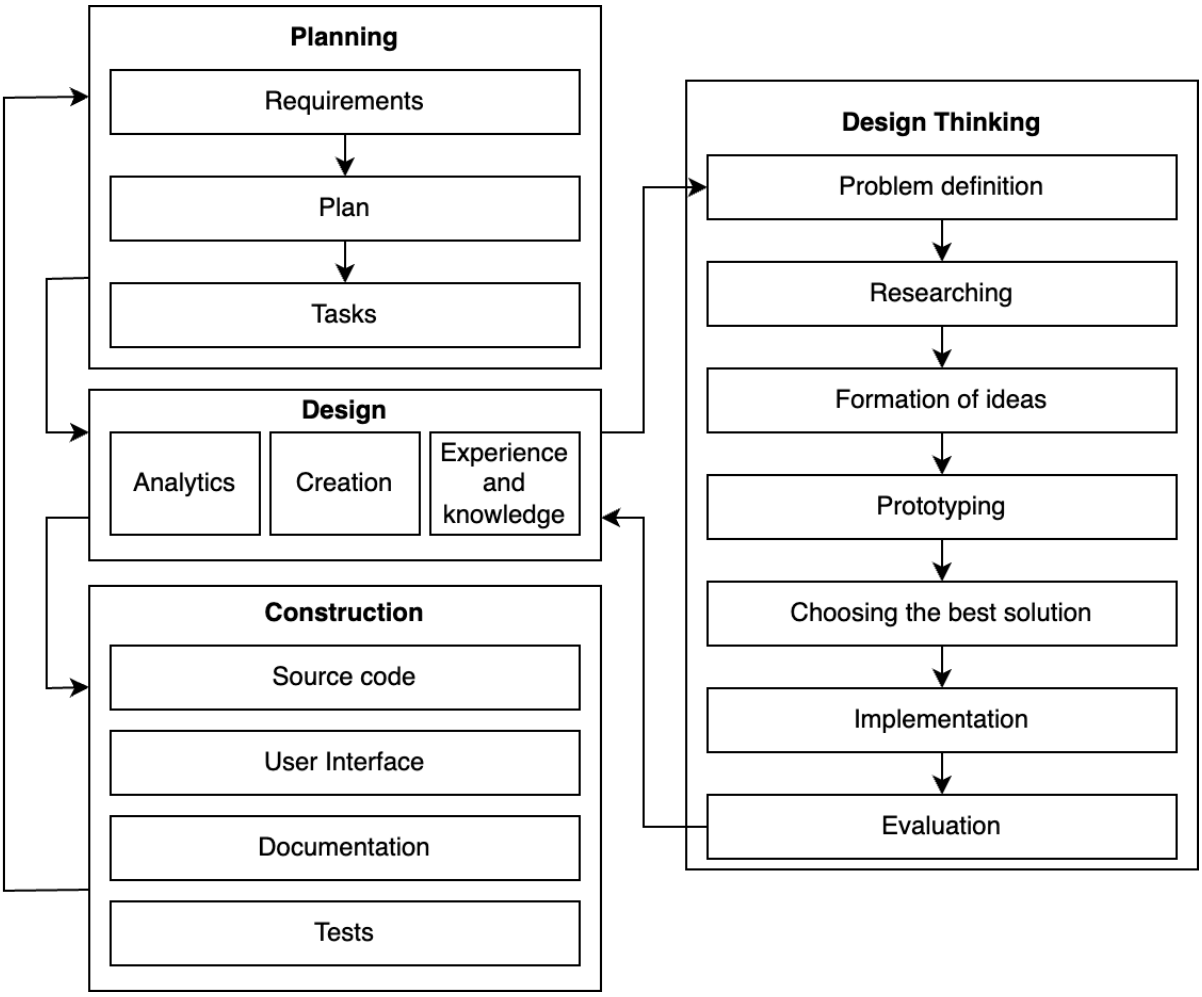

Figure 1. Flexible development iteration structure.

As you can see from Figure 1, the quality of the planning and design stages affects the quality of the software design stage:

1. The result of the planning stage depends on the quality of the analysis of functional and non-functional requirements [4], as well as on the quality of management decisions. We can represent management decisions as a set of tasks for developers and as a set of team management decisions. The project manager at the planning stage must consider the limitations of the resources, the limitations of the real world (domain), and the quality requirements.
2. The result of the design stage depends on the planning stage and the qualifications of designers. Moreover, the design stage is a creative process in terms of the DT methodology, which requires the development of automated CS generation tools [2,3].

As you can see from the review of publications about the study, the use of methods of intellectual analysis and knowledge engineering makes it possible to automate the design stage based on the formalization of the experience of previous projects.

3.1. Knowledge Base Model for Formalizing of the Experience of Previous Projects

The following things influence the development process of a software system [15]:

- A requirement to meet stakeholder needs;
- Project constraints;
- Quality attributes;
- Architectural decisions;
- Aims and goals;
- Stakeholder expectations, etc.

Thus, the development of the software system must be considered within the life cycle, the requirements, and the set of adopted design decisions. We consider the architecture of the software system as a set of representations of this architecture: a business representation, physical representation, and technical representation. The AD comprises design artifacts. The design artefact is the most primitive construction of an AD.

The AD is formed in the process of software system architecting. The AD can also be obtained by reworking the architecture description of previous projects [15].

The AD can be used within the life cycle of a software system in the following ways:

- As a basis for the design and construction processes of a software system;
- As a basis for the analysis and evaluation of alternative implementations of an AD;
- As documentation in the development and maintenance processes of a software system;
- To document significant aspects of a software system;
- As input to automated tools for modeling, system simulatio,n and analysis;
- To define a group of software systems that have common properties (for example, architectural styles, reference architectures, and product line architectures);
- For communication between the teams involved in software system development;
- To provide communication between customers and developers;
- To document the characteristics, properties, and features of a software system;
- As a basis for planning the transition from a legacy architecture to a new one;
- As a guide to operational and infrastructure support and the configuration management of a software system;
- To support system planning and activities related to timelines and budgets;
- As a basis for audits, analysis, and evaluation of a software system;
- As a basis for the analysis and evaluation of alternative architectures;
- For reusing the architectural knowledge through points of view, patterns, and styles;
- To educate stakeholders on best practices for architecting and development.

The authors of the following papers [22,23] describe that ontologies can be used in architecting instead of traditional software system modeling languages (such as UML) since ontologies allow us to control the logical integrity and consistency of the resulting model. However, the existing methods of forming ontologies to support and automate the designing of software systems require the involvement of domain experts and specialists in knowledge engineering. The manual creation of ontologies requires significant time costs.

The main difficulty in creating knowledge bases to support the software systems development lies in the need to unify design artifacts. The formats and methods for storing design artifacts are different, which makes it difficult to analyze and use them in new software systems' development.

Considering that the specifics of the design knowledge in an AD lead to the need to form a knowledge base with a special structure, the knowledge base must include a set of representations for describing the following [15]:

- The concepts of a domain;

- The features of design artifacts formalized as knowledge base fragments;
- The features of the development process as the main stages of a software system life cycle;
- Sets of semantic relations between knowledge base entities;
- Interpretation functions.

Ontologies are based on different description logics (DLs). DLs can guarantee the logical integrity and consistency of the ontology. DLs have decidability and a relatively low computational complexity. These features of DL provide a compromise between expressiveness and decidability. The Web Ontology Language (OWL) is a family of knowledge representation languages for authoring ontologies.

The main component of an OWL 2 ontology is a set of axioms — statements that say what is true in the domain [24,25]. OWL 2 provides an extensive set of axioms, all of which extend the axiom class in the structural specification. Axioms in OWL 2 can be declarations, axioms about classes, axioms about object or data properties, datatype definitions, keys, assertions (sometimes also called facts), and axioms about annotations.

We use the following DL axioms to describe the terminology of the proposed knowledge base [24]:

- $\top$ is a special class with every individual as an instance (top);
- $\bot$ is an empty class (bottom);
- $A \sqsubseteq B$ is the class inclusion axiom (A is a subclass of B);
- $A \sqcap B \sqsubseteq \bot$ is the disjoint classes axiom;
- $A \sqcap B, R_1.A \sqcap R_2.B$ is the intersection or conjunction of axioms (classes or roles);
- $\forall R.A$ is the universal restriction axiom;
- $\exists R.A$ is the existential restriction axiom;
- $\leq 1R.A$ is the functional roles axiom;
- $Inv(R_1) \sqsubseteq R_2$ is the inverse roles axiom;
- $R_1 \circ R_2 \sqsubseteq R_3$ is the transitive roles axiom;
- $\exists R.Self \sqsubseteq \bot$ is the irreflexive roles axiom.

We represent the knowledge base model for formalizing the experience of previous projects using the following definition:

$$B = \{B_1, B_2, \ldots, B_i, \ldots, B_n\},$$

where B_i is an i-th indexed software project that we can define as follows:

$$B_i = \langle L^B, P^B, T^B, D^B, W^B, R^B \rangle, \quad (1)$$

where L^B is the representation of the development process. The representation of the development process allows for the consideration of the specifics of the project life cycle. In addition, this view can help a project manager to evaluate the impact of management decisions on the software project dynamics, for example, how did an increase in the number of developers affect development activity or project quality, etc.

P^B is the representation of the software project structure (directories and files). The representation of the software project structure allows for the obtainment of information about the structure of the files and directories of a software project to classify files into the following types: the source code, documentation, tools to build/compile the code, tests, an additional data directory, external dependencies (libraries), directory with binaries, etc. This information allows for the use of the necessary analysis methods for files with different types, as well as considering the structure of the project when extracting the design patterns.

T^B is the representation of the software project environment (a set of a technology components). The representation of the software project environment allows for the extraction of information about the software system environment: dependencies (libraries), external components (services), runtime environments, etc. In addition, this representation allows for the consideration of various architectures styles and design patterns that developers

used in a project. Information about the environment is very important because an incorrectly configured environment can cause errors in the software system. The environment information also allows for the researching of only those completed projects that meet the requirements of the current project.

D^B is the representation of domain features. The representation of domain features allows for the definition of a problem area and the main use cases of a software project. W^B is the representation of the linguistic environment (concepts and terms). The linguistic environment allows for the equation of objects that have different names but have the same semantics, for example, employee and staff, development and construction, etc.

R^B is the set of relations between knowledge base representations. We will discuss these relationships next.

Let us consider in more detail the components of the project representation in the knowledge base context (Equation 1).

The common terminology of the knowledge base is:

$$\top \equiv \exists hasName.String \sqcap \forall hasName.String \sqcap \leq 1 hasName.String,$$
$$Project \sqsubseteq \top,$$

where *hasName* is a functional role common to all knowledge base classes. The *hasName* role allows us to specify the name of a class individual (object);

Project is a class for describing a software project.

The terminology for the representation of the project development process L^B can be described as the following axioms (Figure 2):

- A set of classes for describing the following:
  - Milestone—*Milestone*;
  - Issue—*Issue*;
  - Merge/pull request—*Request*;
  - Branch—*Branch*;
  - Commit—*Commit*;
  - Contributor—*Contributor*;
  - File—*File* (this is a part of the representation of the software project structure P^B).
- The *Project* class has the following:
  - The *hasMilestone*, *hasRequest*, *hasIssue*, and *hasBranch* roles to specify ties between a project and a set of its milestones, merge/pull requests, issues, and branches;
  - The *hasCommits* and *hasContributors* transitive roles to define ties between a project and a set of its commits and contributors;
  - The *hasDescription* functional role to specify a tie between a project and its description:

$$\begin{aligned} Project \sqsubseteq\ & \forall hasMilestone.Milestone \sqcap \forall hasRequest.Request \sqcap \forall hasIssue.Issue \sqcap \\ & \sqcap \forall hasBranch.Branch \sqcap \exists hasBranch.Branch \sqcap \\ & \sqcap \exists hasCommits.Commit \sqcap \forall hasCommits.Commit \sqcap \\ & \sqcap \exists hasContributors.Contributor \sqcap \forall hasContributors.Contributor \sqcap \\ & \sqcap \exists hasDescription.String \sqcap \forall hasDescription.String \sqcap \\ & \sqcap \leq 1\ hasDescription.String \\ hasBranch \circ hasCommit \sqsubseteq\ & hasCommits \\ hasCommits \circ hasContributor \sqsubseteq\ & hasContributors. \end{aligned}$$

- The *Milestone* class has the following:
  - The *hasRequest*, *hasIssue*, and *hasComment* roles to specify ties between a milestone and a set of its merge/pull requests, issues, and comments;
  - The *hasDescription* functional role to specify a tie between a milestone and its description;

– The $fromProject$ inverse functional role to define a tie between a milestone and its project:

$$\begin{aligned}
&Milestone \sqsubseteq \forall hasRequest.Request \sqcap \forall hasIssue.Issue \sqcap \forall hasComment.String \sqcap \\
&\quad \sqcap \exists hasDescription.String \sqcap \forall hasDescription.String \sqcap \\
&\quad \sqcap \leq 1 hasDescription.String \sqcap \\
&\quad \sqcap \exists fromProject.Project \sqcap \forall fromProject.Project \sqcap \leq 1\ fromProject.Project \\
&Inv(hasMilestone) \sqsubseteq fromProject.
\end{aligned}$$

- The *Issue* class has the following:
  – The *hasContributor* and *hasComment* roles to specify ties between an issue and a set of its contributors and comments;
  – The *hasDescription* functional role to specify a tie between an issue and its description;
  – The $fromMilestone$, $fromRequest$, $fromBranch$, and $fromProject$ inverse functional roles to define ties between an issue and its milestone, merge/pull request, branch, and project:

$$\begin{aligned}
&Issue \sqsubseteq \exists hasContributor.Contributor \sqcap \forall hasContributor.Contributor \sqcap \\
&\quad \sqcap \forall hasComment.String \sqcap \\
&\quad \sqcap \exists hasDescription.String \sqcap \forall hasDescription.String \sqcap \\
&\quad \sqcap \leq 1\ hasDescription.String \\
&\quad \sqcap \forall fromMilestone.Milestone \sqcap \leq 1\ fromMilestone.Milestone \\
&\quad \sqcap \forall fromRequest.Request \sqcap \leq 1\ fromRequest.Request \\
&\quad \sqcap \exists fromBranch.Branch \sqcap \forall fromBranch.Branch \sqcap \leq 1\ fromBranch.Branch \\
&\quad \sqcap \exists fromProject.Project \sqcap \forall fromProject.Project \sqcap \leq 1\ fromProject.Project \\
&Inv(hasIssue) \sqsubseteq fromMilestone \sqcap fromRequest \sqcap fromBranch \sqcap fromProject.
\end{aligned}$$

- The *Request* class has the following:
  – The *hasIssue* and *hasComment* roles to specify ties between a merge/pull request and a set of its issues and comments;
  – The *hasDescription* functional role to specify a tie between a merge/pull request and its description;
  – The $fromMilestone$, $fromBranch$, and $fromProject$ inverse functional roles to define ties between a merge/pull request and its milestone, branch, and project;
  – The *hasCommits* and *hasContributors* transitive roles to define ties between a merge/pull request and a set of its commits and contributors:

$$\begin{aligned}
&Request \sqsubseteq \forall hasIssue.Issue \sqcap \forall hasComment.String \sqcap \\
&\quad \sqcap \exists hasDescription.String \sqcap \forall hasDescription.String \sqcap \\
&\quad \sqcap \leq 1\ hasDescription.String \\
&\quad \sqcap \forall fromMilestone.Milestone \sqcap \leq 1\ fromMilestone.Milestone \\
&\quad \sqcap \exists fromBranch.Branch \sqcap \forall fromBranch.Branch \sqcap \leq 1\ fromBranch.Branch \\
&\quad \sqcap \exists fromProject.Project \sqcap \forall fromProject.Project \sqcap \leq 1\ fromProject.Project \\
&\quad \sqcap \exists hasCommits.Commit \sqcap \forall hasCommits.Commit \sqcap \\
&\quad \sqcap \exists hasContributors.Contributor \sqcap \forall hasContributors.Contributor \sqcap \\
&Inv(hasRequest) \sqsubseteq fromMilestone \sqcap fromBranch \sqcap\ fromProject \\
&hasBranch \circ hasCommit \sqsubseteq\ hasCommits \\
&hasCommits \circ hasContributor \sqsubseteq hasContributors.
\end{aligned}$$

- The *Branch* class has the following:

– The *hasCommit*, *hasRequest*, and *hasIssue* roles to specify ties between a branch and a set of its commits, merge/pull requests, and issues;
– The $fromProject$ inverse functional role to define a tie between a branch and its project:

$$\begin{aligned} Branch \sqsubseteq\ & \exists hasCommit.Commit \sqcap \forall hasCommit.Commit \sqcap \\ & \sqcap \forall hasRequest.Request \sqcap \forall hasIssue.Issue \sqcap \\ & \sqcap \exists fromProject.Project \sqcap \forall fromProject.Project \sqcap \leq 1\ fromProject.Project \\ Inv(hasBranch) \sqsubseteq\ & fromProject. \end{aligned}$$

- The *Commit* class has the following:
  – The *hasContributor* functional role to specify a tie between a commit and its contributor;
  – The *hasComment* and $modifyFile$ roles to specify ties between a commit and a set of its comments and modified files;
  – The $fromBranch$ and $fromProject$ inverse functional roles to define ties between a commit and its branch and project;
  – The *hasMessage* and *hasDate* functional roles to specify a tie between a commit and its description and date:

$$\begin{aligned} Commit \sqsubseteq\ & \exists hasContributor.Contributor \sqcap \forall hasContributor.Contributor \sqcap \\ & \sqcap \leq 1 hasContributor.Contributor \sqcap \forall hasComment.String \sqcap \\ & \sqcap \exists modifyFile.File \sqcap \forall modifyFile.File \sqcap \\ & \sqcap \exists fromBranch.Branch \sqcap \forall fromBranch.Branch \sqcap \leq 1\ fromBranch.Branch \\ & \sqcap \exists fromProject.Project \sqcap \forall fromProject.Project \sqcap \leq 1\ fromProject.Project \\ & \sqcap \exists hasMessage.String \sqcap \forall hasMessage.String \sqcap \leq 1 hasMessage.String \sqcap \\ & \sqcap \exists hasDate.Date \sqcap \forall hasDate.Date \sqcap \leq 1\ hasDate.Date \\ Inv(hasCommit) \sqsubseteq\ & fromBranch \sqcap fromProject. \end{aligned}$$

- The *Contributor* class has the following:
  – The *hasIssue*, *hasCommit*, and *hasRequests* roles to specify ties between a contributor and a set of its issues, commits, and requests;
  – The $fromProject$ inverse functional role to define a tie between a contributor and its project:

$$\begin{aligned} Contributor \sqsubseteq\ & \forall hasIssue.Issue \sqcap \forall hasCommit.Commit \sqcap hasRequest.Request \sqcap \\ & \sqcap \exists fromProject.Project \sqcap \forall fromProject.Project \sqcap \leq 1\ fromProject.Project \\ Inv(hasContributor) \sqsubseteq\ & fromProject. \end{aligned}$$

As you can see in Figure 2, dashed lines are used to illustrate some entities and relationships. Such entities and relationships may not be contained in an indexed repository and therefore may not be represented in the knowledge base.

We describe the P^B representation of a software project structure as the following axioms:

- A set of classes for describing the following:
  – Directory—*Directory*;
  – File—*File*;
  – Commit—*Commit* (is a part of the representation of the project development process L^B);

- The *Commit* class has the *hasSourceDirectory* and *hasBuildFile* functional roles to define ties between a commit and its source directory and build file:

$$\begin{aligned} Commit \sqsubseteq\ & \exists hasSourceDirectory.Directory \sqcap \forall hasSourceDirectory.Directory \sqcap \\ & \sqcap \leq 1 hasSourceDirectory.Directory \sqcap \\ & \sqcap \exists hasBuildFile.File \sqcap \forall hasBuildFile.File \sqcap \leq 1 hasBuildFile.File. \end{aligned}$$

- The *Directory* class has the following:
  - The *include* irreflexive role to specify ties between a directory and a set of its subdirectories;
  - The *includeFile* role to define a ties between a directory and a set of its files:

$$\begin{aligned} & Directory \sqsubseteq \forall include.Directory \sqcap \forall\ includeFile.File \\ & \exists include.Self \sqsubseteq \bot. \end{aligned}$$

Let us see the representation of a software project environment T^B as the following axioms:

- A set of classes for describing the following:
  - Architecture styles or design patterns—*Arch*. The *Arch* set is defined using an enumerated class. The enumerated class contains a list of architectural styles and design patterns for which we develop the search and analysis method;
  - Third-party dependencies of a software project (libraries, frameworks, external services, database management systems, etc.)—*Dependency*;
  - File—*File* (this is a part of the representation of the software project structure P^B):

$$Arch \equiv \{MVC, DependencyInjection, Facade\}.$$

- The *File* class has the *hasArch* and *hasDependency* roles to specify ties between a file and a set of its architecture styles or design patterns and third-party dependencies:

$$File \sqsubseteq \forall hasArch.Arch \sqcap \forall hasDependency.Dependency.$$

- The *Dependency* class has the *hasGroup*, *hasName*, and *hasVersion* functional roles to define the dependency properties (group, name, and version):

$$\begin{aligned} Dependency \sqsubseteq\ & \exists hasGroup.String \sqcap \forall hasGroup.String \sqcap \leq 1 hasGroup.String \sqcap \\ & \sqcap \exists hasName.String \sqcap \forall hasName.String \sqcap \leq 1 hasName.String \sqcap \\ & \sqcap \exists hasVersion.String \sqcap \forall hasVersion.String \sqcap \leq 1 hasVersion.String. \end{aligned}$$

We describe the representation of the domain features D^B by the following axioms:

- A set of classes for describing the following:
  - Domain entities—*Entity*;
  - Domain business processes—*Process*;
  - File—*File* (this is a part of the representation of the software project structure P^B).
- The *File* class has the *hasEntity* and *hasBusinessMethod* roles to specify ties between a file and a set of its entities and business processes:

$$File \sqsubseteq \forall hasEntity.Entity \sqcap hasBusinessMethod.Process.$$

- The *Entity* class has the *hasProcess* role to define a tie between an entity and a set of its business processes:

$$Entity \sqsubseteq \forall hasProcess.Process.$$

We can represent the representation of the linguistic environment W^B as the following axioms:

- A set of classes for describing the following:
  - Domain concepts—*Concept*. Concepts describe various entities and processes of some domain;
  - Terms that describe a domain concept—*Term*. Terms allow us to associate the names of various software project objects with the concepts of the linguistic environment;
  - Domain entities—*Entity* (this is a part of the representation of domain features D^B);
  - Domain business processes *Process* (this is a part of the representation of domain features D^B).
- The *Entity* and *Process* classes have the *hasConcept* functional role to specify a tie between an entity or process and its concept:

$$Entity \sqsubseteq \forall hasConcept.Concept \sqcap \leq 1\ hasConcept.Concept$$
$$Process \sqsubseteq \forall hasConcept.Concept \sqcap \leq 1 hasConcept.Concept.$$

- The *Concept* class has the *hasTerm* role to define ties between a concept and its terms:

$$Concept \sqsubseteq \forall hasTerm.Term.$$

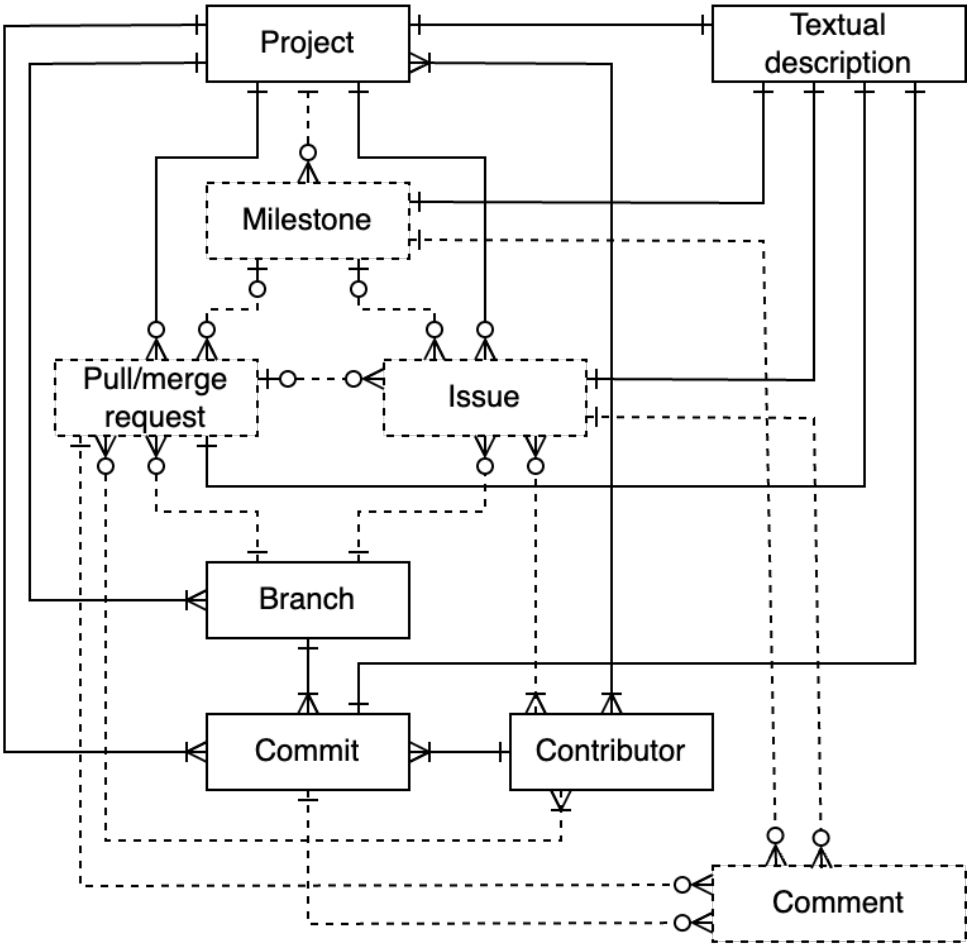

Figure 2. ER diagram of the L^B representation of the software development process.

The classes of all the representations of the proposed knowledge base are declared as disjoint:

$$Project \sqcap Milestone \sqcap Issue \sqcap Request \sqcap Branch \sqcap Commit \sqcap Contributor \sqcap$$
$$\sqcap Directory \sqcap File \sqcap Arch \sqcap Dependency \sqcap Entity \sqcap Process \sqcap Concept \sqcap Term \sqsubseteq \bot.$$

It is necessary to develop a function to map the project of a software system to the model of the proposed knowledge base. That function can be represented as the following definition:

$$F^B : URL \to B_i,$$

where URL is a unified resource locator of a software project repository on the Internet; B_i is the representation of a software project as a fragment of the proposed knowledge base.

The Section 3.2 describes the function F^B. At the moment, we implement the function F^B algorithmically. Currently, the function F^B supports projects in the Java language or the Spring framework only. In the future, we plan to form a metamodel to unify the behavior of the indexer that implements the function F^B. The metamodel will make it possible to implement universal (in most cases) algorithms for formalizing projects for most structural and object-oriented programming languages.

3.2. Formalizing the Experience of Previous Projects

It is necessary to implement the mapping function for formalizing the experience of previous projects as fragments of the proposed knowledge base. The mapping function is based on an algorithm that comprises the following steps.

We consider the ng-tracker task tracker [26] as an example of data for indexing for the knowledge base population. This project is written in Java with the Spring Boot framework. For compactness, we consider only the 'ru.ulstu.conference' package and associated with that package issue #57 ('Creating classes for the Conference module') from the milestone #681923 ('Conferences').

Step 1.Extraction of the representation of the project development process L^B.

Representation of the project development process L^B is formed from two sources [27]:

- The project-hosting API (GitLab, GitHub, etc.);
- The project Git repository.

The Git repository of a project is the preferred source, because it is more stable in terms of API changing and is always available to use. However, it is impossible to extract information about the stages of the development process (milestones, issues, merge/pull requests) only from a git repository.

We developed a software module to work with the GitLab REST API [28]. The following HTTP requests are used for interactions between the module and the GitLab API:

1. GET https://gitlab.com/api/v4/projects/romanov73%2Fng-tracker/milestones
   This HTTP-request is used to obtain the list of the ng-tracker project milestones.
2. GET https://gitlab.com/api/v4/projects/romanov73%2Fng-tracker/milestones/681923/issues
   This HTTP-request is used to obtain the issues of the milestone #681923 ('Conferences').
3. GET https://gitlab.com/api/v4/projects/romanov73%2Fng-tracker/issues/57/related_merge_requests
   This HTTP-request is used to obtain the merge request of the issue #57 ('Creating classes for the Conference module').

After that, the JGit library [29] is used to extract the following information from each commit:

- The SHA hash;
- The date;
- The message (description);

- The branch;
- The contributor;
- The diff (changes).

If there are no milestones and/or issues in the project, then the module extracts only information about commits using the JGit library.

Figure 3 demonstrates the fragment of the L^B representation extracted from the issue #57 of the milestone #681923 of the ng-tracker project.

All examples of representations of the knowledge base are illustrative. In fact, a primary key is generated for each entity, and all relationships between entities are formed based on these keys.

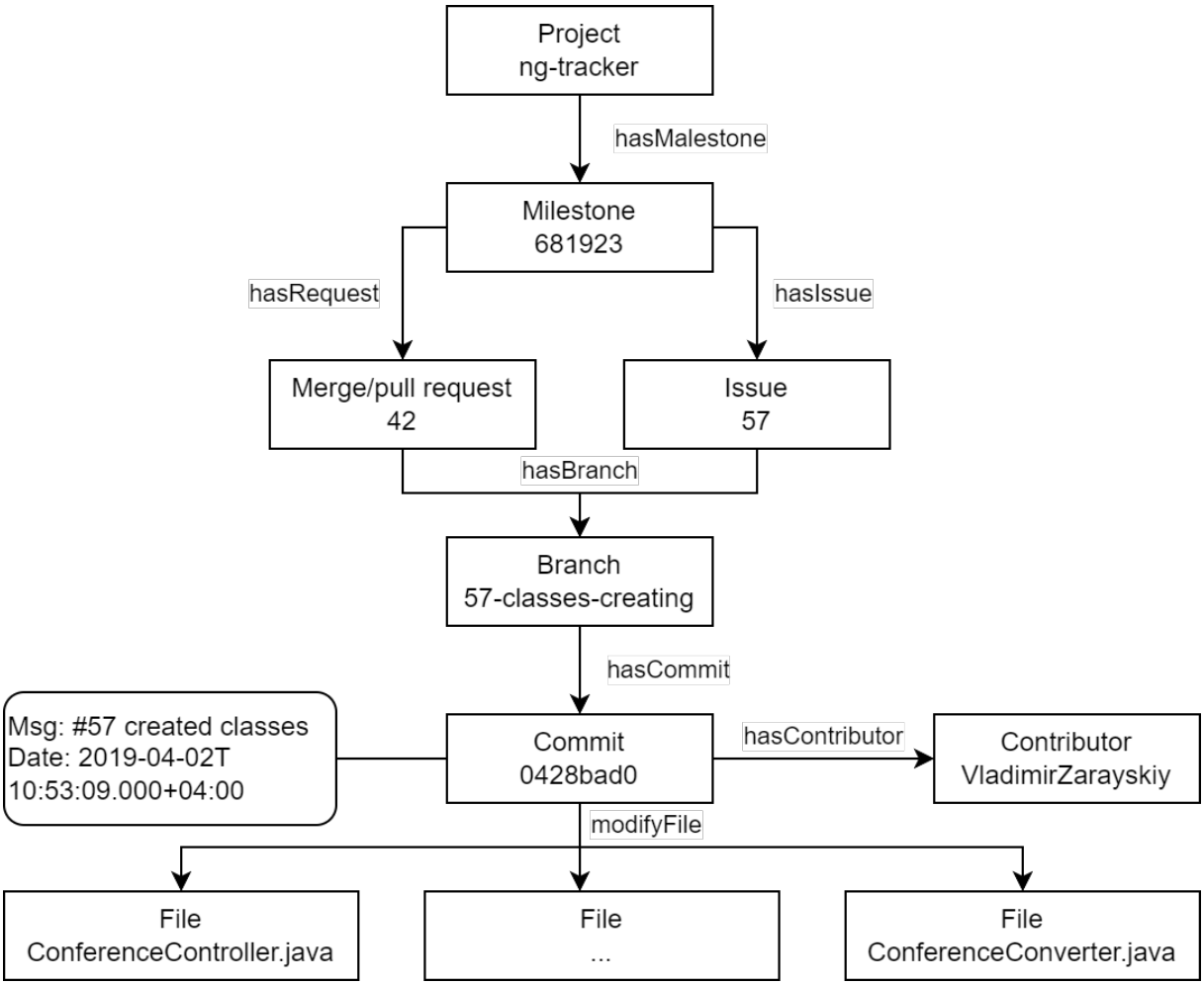

Figure 3. Example of the L^B representation fragment.

Step 2. Extraction of the representation of a software project structure P^B.

The process of the structural analysis of a software project directory is used to create the representation of a software project structure P^B [30]. The indexer module contains typical paths for finding the source code root path and various environment files for different programming languages and build systems. Figure 4 demonstrates the fragment of the P^B representation extracted from the ng-tracker project.

As you can see from Figure 4, the following entities of the P^B representation that are also presented in the L^B representation are marked in gray: the entities with type 'File' and the entity '0428bad0' with type 'Commit'.

Step 3. Extraction of the representation of the software project environment T^B.

To extract the representation of the software project environment T^B, an expert must manually configure the $Arch^T$ and Dep^T components for each programming language,

framework, application type, and other features of the software project environment. We will consider this task and the prospects for its automation in more detail in one of the following papers.

For example, to determine the usage of the MVC pattern for Spring projects, it is necessary to find the *@Controller* annotation on a class and the *@GetMapping*, *@PostMapping*, *@RequestMapping*, etc. annotations for its methods (Listing 1). To determine structural design patterns, it is necessary to consider the project structure information from the P^B representation.

Listing 1. Fragment of the MVC controller of the ng-tracker project.

```
@Controller()
@RequestMapping(value = "/conferences")
@ApiIgnore
public class ConferenceController {
    ...
    @GetMapping("/conferences")
    public void getConferences(ModelMap modelMap) {
        modelMap.put("filteredConferences",
            new ConferenceFilterDto(conferenceService.findAllDto()));
    }
    ...
}
```

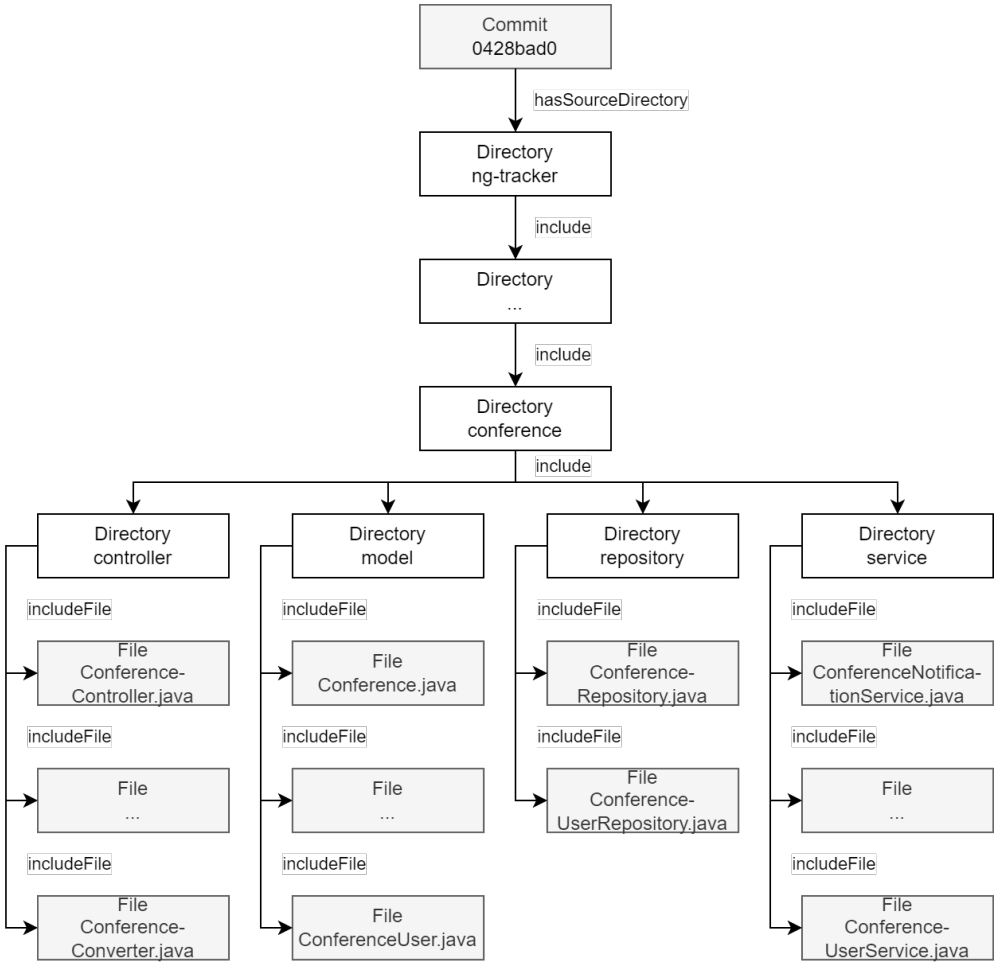

Figure 4. Example of the P^B representation fragment.

The component for extracting the third-party dependencies Dep^T works with the configuration files of build automation tools and extracts the name and version of the

dependency libraries from them. For example, for Java projects the following files are scanned: build.gradle, pom.xml, etc. If a build.gradle file is found, then Gradle is specified as the project build automation tool. Then, the names and versions of the dependency libraries are extracted from the dependencies section of this file (Listing 2).

Listing 2. Fragment of the build.gradle file of the ng-tracker project.

```
dependencies {
    compile group: 'org.springframework.boot',
                name: 'spring-boot-starter-web'
    compile group: 'org.springframework.boot',
                name: 'spring-boot-starter-security'
    ...
    compile group: 'org.postgresql', name: 'postgresql',
                version: '42.2.5'
    compile group: 'org.liquibase', name: 'liquibase-core',
                version: '3.6.3'
    ...
}
```

Figure 5a demonstrates the fragment of the T^B representation with a set of architectural styles extracted from the ng-tracker project, and Figure 5b shows a set of the third-party dependencies.

In Figure 5, we mark entities that were used in other representations (Figures 3 and 4) in gray.

Step 4. Extraction of the representation of domain features D^B.

The representation of domain features D^B is formed by searching in the source tree of the project for classes that describe the data models and business logic [27]. Each programming language and framework requires an indexer configuration to find the corresponding language/framework operators and constructions. For example, for Spring Boot projects, classes that describe data models are marked with the @Entity annotation (Listing 3), and business logic classes are marked with the @Service annotation (Listing 4).

Listing 3. Fragment of the entity class of the ng-tracker project.

```
@Entity
@Table(name = "conference")
@DiscriminatorValue("CONFERENCE")
public class Conference extends BaseEntity
        implements UserActivity, EventSource {
        ...
}
```

Listing 4. Fragment of the business logic class of the ng-tracker project

```
@Service
public class ConferenceService extends BaseService {
        ...
        public List<Conference> findAll() {
                return conferenceRepository.findAll(
                        new Sort(Sort.Direction.DESC, "beginDate"));
        }
        ...
}
```

Moreover, similar classes can be found by searching for specific keywords in class names and their paths or by analyzing class methods. When the classes are found, the names of the entity classes are determined as the entities of the D representation. After that, public methods in which entities are used as arguments or return values are extracted from the business logic classes.

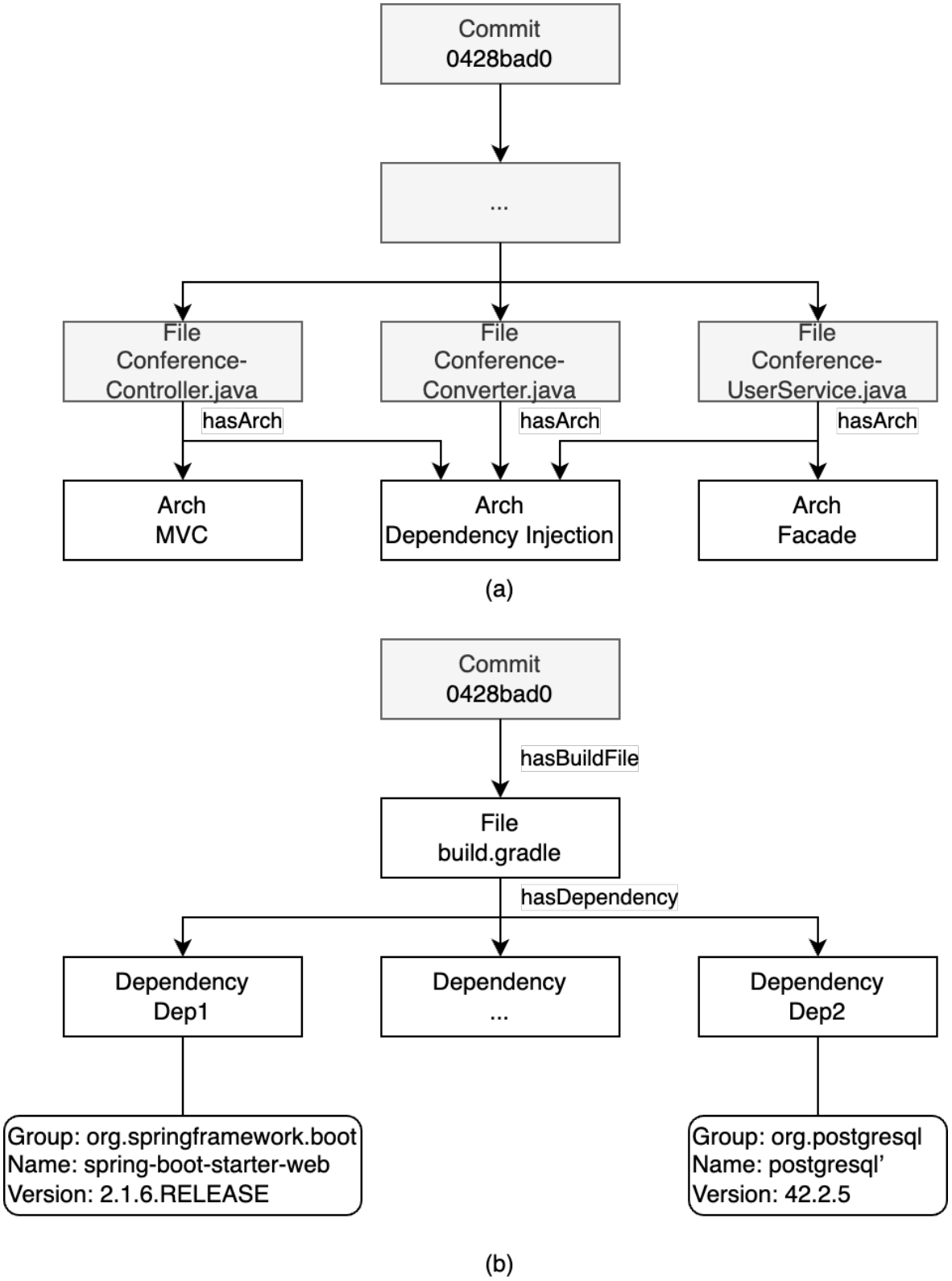

Figure 5. Example of the T^B representation fragment: (**a**) a set of architectural styles; (**b**) a set of the third-party dependencies.

Figure 6 demonstrates the example of the D^B representation fragment.

Step 5. Formation of the the linguistic environment W^B.

We form the linguistic environment by analyzing various text descriptions that are contained in the project repository and represented by terms of natural language using statistical [31] and linguistic [32] analysis methods.

Figure 7 demonstrates the example of the fragment of the W^B representation of the linguistic environment.

As you can see from Figure 7, one entity or business process of the D^B representation can correspond to several concepts of the terminological environment.

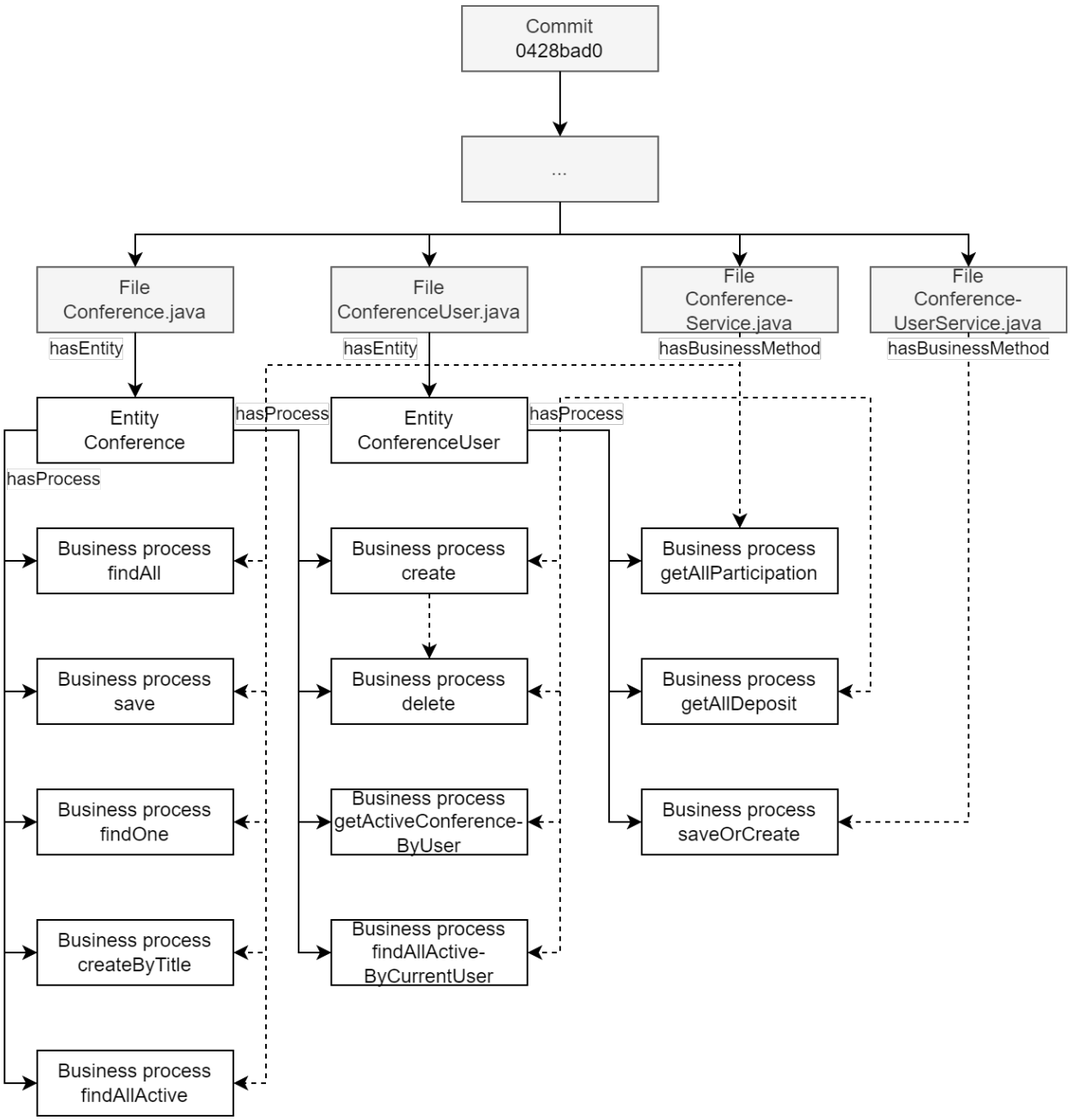

Figure 6. Example of the D^B representation fragment.

Thus, the resulting knowledge base is a source of design experience, based on which it is possible to form methods for automating the building of the CS to support the design stage.

We associate objects of various representations of the proposed knowledge base with commits. Commits contain information about creating, changing, or deleting objects.

The key position of the 'Commit' object allows us to consider the development process of the software project in a dynamic way. The presentation of information about the project in a dynamic way allows for the use of various methods of diagnostic and predictive analytics to improve the quality and efficiency of the decision making of project managers.

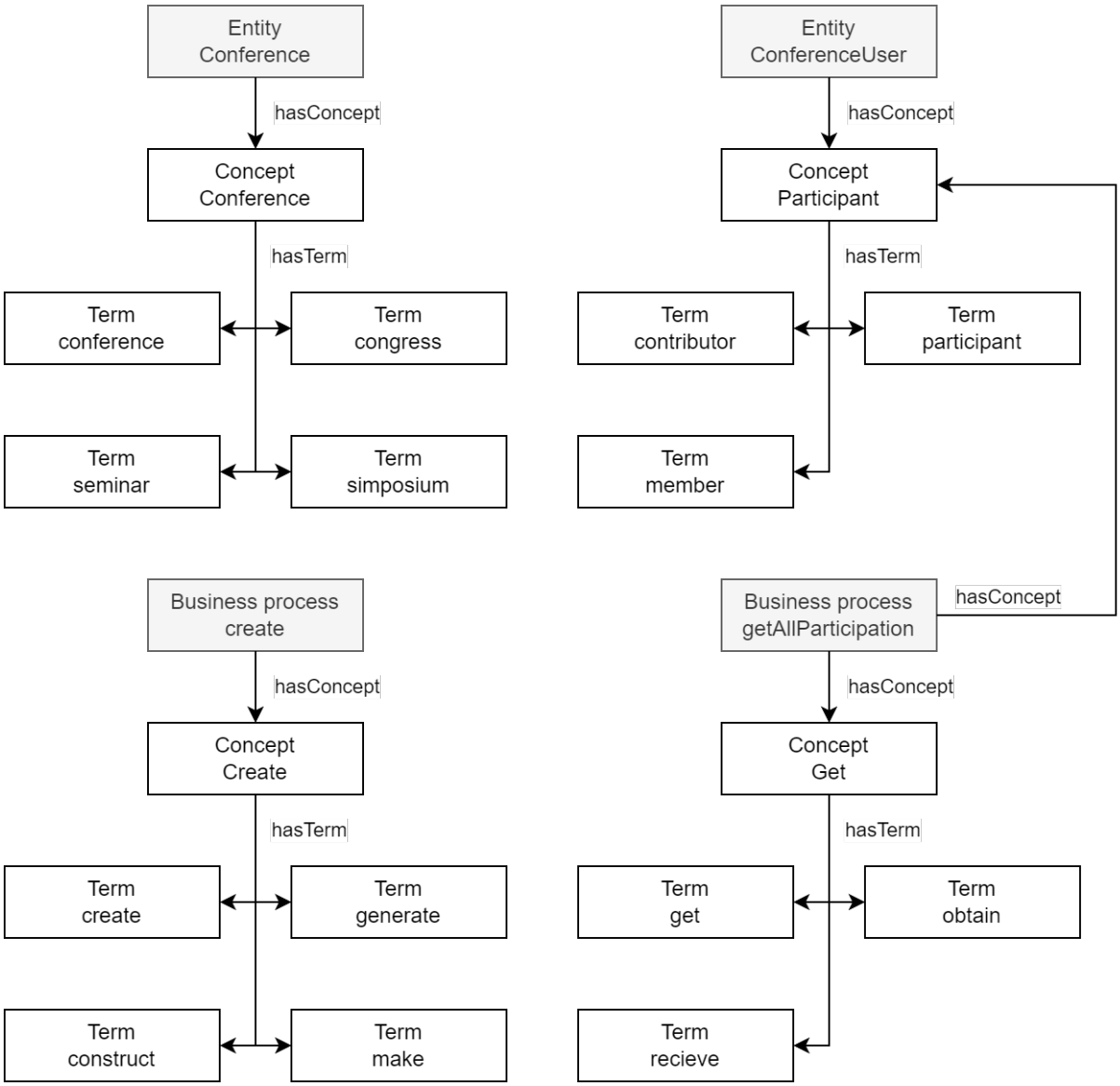

Figure 7. Example of the fragment of the W^B representation of the linguistic environment.

3.3. Diagnostic Analytics Method for Decision Support in Project Management

Using knowledge engineering methods in modeling and analyzing time series makes it possible to consider the limitations and features of some domain. Moreover, knowledge engineering methods allow us to choose the type of model and its parameters to improve the quality of time series analytics [33].

The data source of the proposed diagnostic analytics method is the knowledge base. The model of the knowledge base is presented in the previous section (Equation (1)). In our study, the key entity of the knowledge base is the 'Commit' object. The objects of the L^B representation of the development process (Figure 2) and the objects of the other representations (Figure 8) have ties with the Commit object.

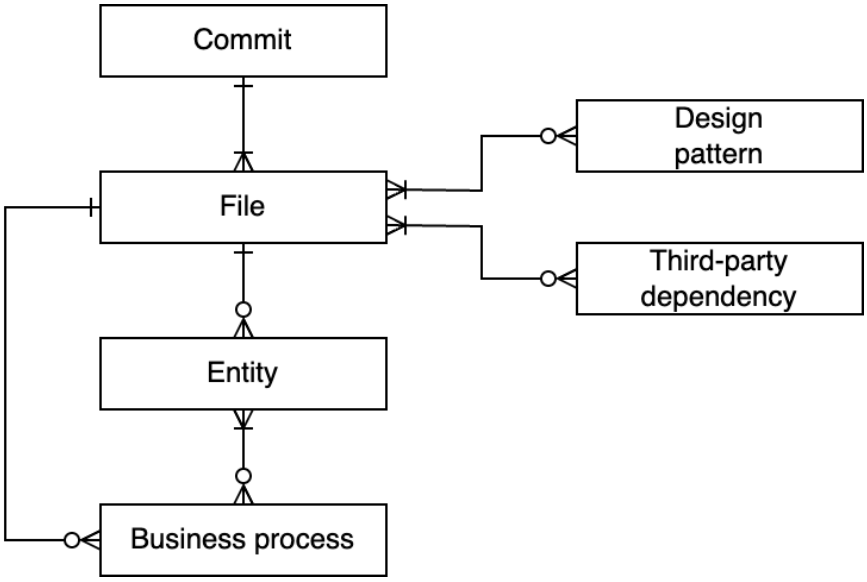

Figure 8. ER diagram for the 'Commit' object.

As you can see from Figure 8, the key position of the 'Commit' object allows us to extract a set of time series of various indicators from the knowledge base based on a set of project commits with the required frequency and discreteness using the aggregation function.

We describe the function for extracting a set of time series from the knowledge base using the following definition:

$$F^{TS}\colon B_i \times Period \to TS,$$

where B_i is a knowledge base fragment for the i repository;
Period are settings for extracting a set of time series: period and discreteness;
$TS = \{TS_1, TS_2, \ldots, TS_i, \ldots, TS_n\}, \forall |TS_i| = m$ is a set of time series of n indicators (one time series for each indicator) with length m extracted from the knowledge base;
TS_i is a time series of the i-th indicator.

Next, we classify a set of time series based on expert rules into time series that have positive and negative impacts on the quality of the development process:

$$F^{Cl}\colon TS \to Dyn,$$

where F^{Cl} is a function for time series classification;
$Dyn = \{TS_1^{pos}, TS_2^{neg}, \ldots, TS_i^{pos}, \ldots, TS_n^{pos}\}$ is a set of time series with a positive or negative dynamic of the project development.

Then, we apply the following function to extract knowledge from time series:

$$F^{St}\colon Dyn \to St,$$

where F^{St} is a function of the time series knowledge extraction. The algorithm of time series knowledge extraction contains the following steps:

1. The evaluation of the indicator value using a set of expert 'if-then' rules. Each rule defines a range of values. The indicator is assigned some linguistic value when an indicator value belongs to a certain interval.
2. The modification of state values based on the mutual influence of indicators on each other. For example, if the number of contributors increases, the state for the number of commits indicator should be changed to a lesser value.

$St = \{\langle i, \{state_1^i, state_2^i, \ldots, state_m^i\}\rangle\}$ is a set of states (trends) for indicators that are presented by the analyzed time series. Linguistic values represent the set of states, for example, few, medium, or many.

The project manager can form recommendations based on a set of states in the planning phase when starting a new iteration of the development process.

4. Results

This section presents the currently implemented functions for the design automation and project management of software systems. Work on the project is in progress, and we are constantly adding new functionality to the software platform.

4.1. Information Retrieval of Software Projects

The popular web services for hosting software projects use information retrieval based on text processing methods that do not consider the specifics of design artifacts [34].

We have developed an information retrieval subsystem that allows for the searching of software projects, considering the specifics of a software project from the knowledge base (Equation (1)). We use the Neo4j graph database management system for organizing a knowledge base. Neo4j has a high speed of query execution [35].

Let us represent the search query as the following definition:

$$Q = \langle L^Q, D^Q \rangle,$$

where L^Q is a set of parameters for information retrieval by the following indicators of the development process from the representation of the project development process L^B:

- The number of contributors;
- The number of commits.

D^Q is a set of parameters for information retrieval by the following domain features from the representation of domain features D^B:

- The name of the entity;
- The name of the business process;
- The number of entities;
- The number of business processes.

We represent the information retrieval function as:

$$F \colon Q \times B \to \tilde{B}, \tilde{B} \subset B,$$

where $\tilde{B}$ is a subset of projects that match query parameters;
B is a set of indexed projects of the knowledge base.

Currently, the user sets query parameters using a special form component that uses a separate input element for each parameter. We form a Cypher query based on the form component data. We use the 'UNION' operator to join all the atoms of the condition in the resulting Cypher query.

The following definition represents the function for calculating the relevance of project $\tilde{B}_i$ to query Q:

$$Rel = \frac{\left|\tilde{B}_i \cap Q\right|}{|Q|},$$

where $\left|\tilde{B}_i \cap Q\right|$ is the number of matching parameters in project B_i and query Q;
$|Q|$ is the number of parameters in query Q.

An index graph is formed and saved in Neo4j for each project in the indexing process. Figure 9 demonstrates the graph for the ng-tracker project.

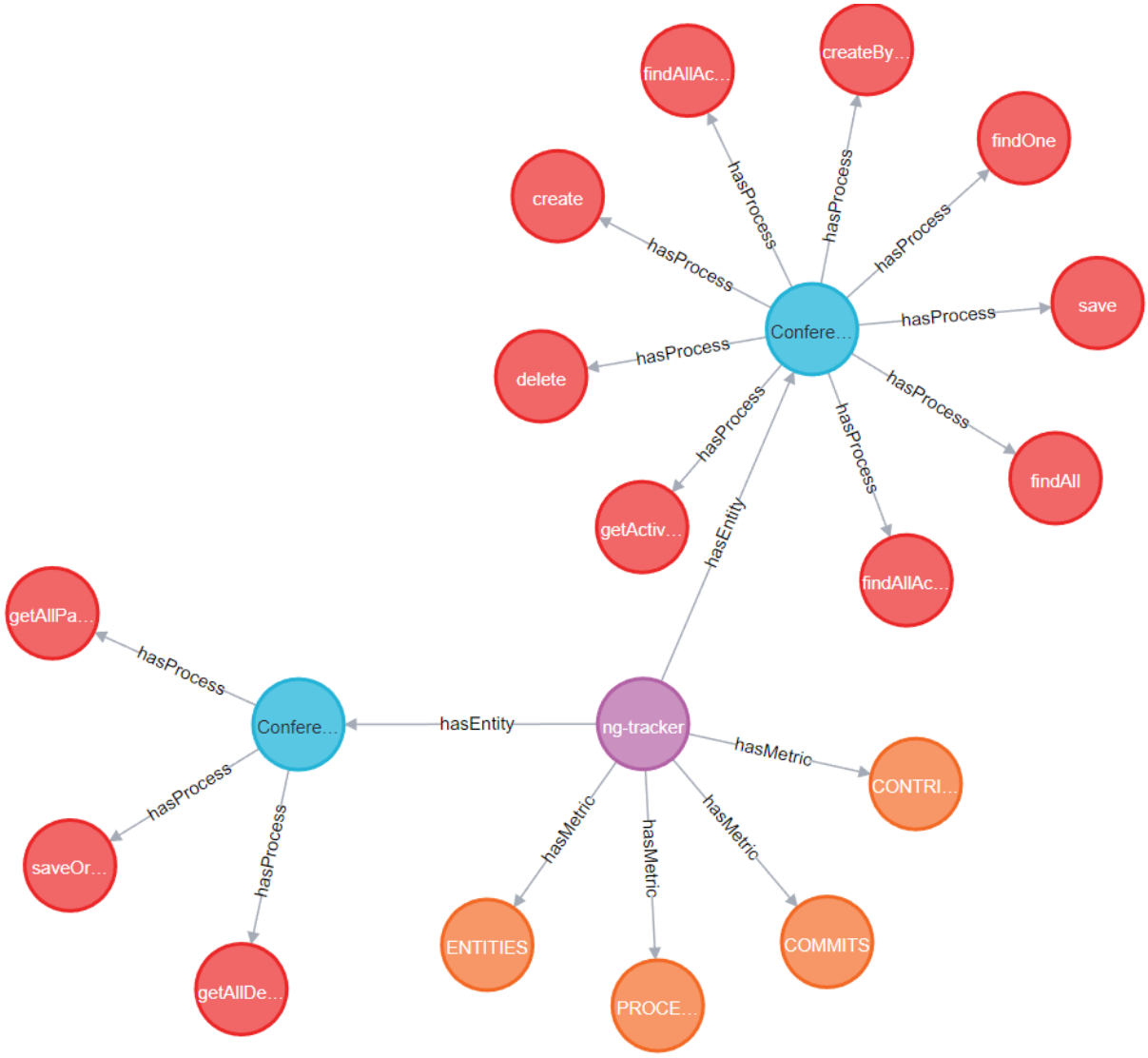

Figure 9. Example of the Neo4j graph for the ng-tracker project.

As you can see from Figure 9, the graph contains the following nodes:

- The 'Project' node.
- The 'Entity' node. These nodes are formed based on the set of project entities from the D^B representation (Figure 6).
- 'Process' node. This type of node is formed based on D^B representation processes associated with a specific entity.
- The 'Metric' node: 'Contributors', 'Commits', 'Entities', and 'Processes'. The values of 'Contributors' and 'Commits' metric nodes are formed based on the aggregation of data by the number of changes and contributors of the L^B representation (Figure 3). The values for 'Entities' and 'Processes' metric nodes are formed based on the number of 'Entity' and 'Process' nodes.

All graph nodes have 'id' and 'name' properties. Metric nodes also have value properties with double types to store the value of the metric.

The following types of relations are used in the graph:

- A 'hasEntity' relation for a 'Project' and an 'Entity' nodes connection;
- A 'hasProcess' relation for an 'Entity' and 'Process' nodes connection;
- A 'hasMetric' relation for a 'Project' and 'Metric' nodes connection.

Listing 5 demonstrates an example of the Cypher query to find projects with relevance calculation and sorting in a descending order of relevance. Such a Cypher query is generated automatically based on the user search parameters.

Listing 5. Example of the Cypher query of the information retrieval subsystem.

```
call {
        match (p:Project)-[:hasEntity]->(e:Entity)
                where toLower(e.name) =~ 'conference'
                return p as project, 1 as weight
        union all
        match (p:Project)-[:hasMetric]->(m:Metric {name: 'COMMITS'})
                where m.value > 1 and m.value < 5
                return p as project, 1 as weight
}
with * return project.name, sum(weight) * 1.0 / 2 as relevant
        order by relevant desc
```

The linguistic environment W^B is used when generating a search query. When the user specifies in a search query the name of an entity or process, it is necessary to find a correspondence between each query term and the terms of the linguistic environment W^B. If the term from the query matches the term of the W^B representation, then the following needs to occur (Figure 7):

1. It needs to transit from the term to concept by the 'hasTerm' relation.
2. Then, it needs to transit to the the entity or business process by the 'hasConcept' relation.

If we could obtain the entity or business process, then the corresponding term in the search query is replaced with the name of the entity or business process.

We plan to add support for additional search parameters for the information retrieval subsystem. We also plan to use fuzzy logic methods to represent quantitative data as linguistic values. For example: a small project, an average size of a development team, etc.

Thus, the proposed information retrieval subsystem allows for the automation of the research phase at the design stage by reducing the time costs and search space.

4.2. Generating Use Case Diagrams in UML Notation

The platform currently implements the function for generating use case diagrams in UML notation to automate the building of the CS.

We represent a use case diagram in UML notation as the following definition:

$$UCD = \langle A^{UCD}, S^{UCD}, P^{UCD}, R^{UCD} \rangle,$$

where A^{UCD} is a set of actors that perform certain roles in a given system;
S^{UCD} is a set of system boundaries that define the limits of the system;
P^{UCD} is a set of use cases that represent a business functionality;
and $R^{UCD} = \langle R_I^{UCD}, R_E^{UCD}, R_G^{UCD}, R_A^{UCD} \rangle$ is a set of relations:

- R_I^{UCD} is an include relationship, a use case that includes the functionality described in another use case as a part of its business process flow;
- R_E^{UCD} is an extend relationship, where the child use case adds to the existing functionality and characteristics of the parent use case;
- R_G^{UCD} is a generalization relationship, a parent–child relationship between use cases;
- R_A^{UCD} is an association relationship, a relationship between actors and use cases.

At the moment, we generate diagrams based only on the hierarchy of the entities and business processes of the representation of domain features D^B (Equation (1)):

$$F^{UCD} \colon D^{B_i} \to UCD.$$

The proposed algorithm creates a use case diagram as a set of commands for the PlantUML system [36]. Now only actors, use cases, and association and include relationships are formed in the resulting use case diagram.

The entities and business processes of the D^B representation (Figure 6) of the ng-tracker project are used to generate a use case diagram.

The use case diagram generation algorithm contains the following steps:

1. Create an actor with the name 'User': A_1^{UCD}:

```
:User:
```

2. Create a root use case: P_1^{UCD}. Specify the name of the project as the name for a root use case:

```
(ng-tracker)
```

3. Connect a root use case with an association relation with an actor: $P_1^{UCD} R_A^{UCD} A_1^{UCD}$:

```
:User: - (ng-tracker)
```

4. Form a use case for each entity and connect it with an inclusion relation with the root use case:

$$\forall E_i^D \rightarrow P_{1i}^{UCD}, E_i^D \in D^B, P_{1i}^{UCD} R_I^{UCD} P_1^{UCD}.$$

Specify the name of an entity as the name of a use case:

```
(ng-tracker)..>(Conference):include
(ng-tracker)..>(ConferenceUser):include
```

5. Obtain a list of business processes (P^{E_i}) for each entity. Create a use case for each business process from P^{E_i} and connect it with an inclusion relation with a parent use case (entity E_i):

$$\forall P_j^{E_i} \rightarrow P_{ij}^{UCD}, P_j^{E_i} \in D^B, P_{ij}^{UCD} R_I^{UCD} P_{1i}^{UCD}.$$

Specify the name of a business process as the name of a use case:

```
(Conference)..>(findAll):include
(Conference)..>(save):include
(Conference)..>(findOne):include
(Conference)..>(createByTitle):include
(Conference)..>(findAllActive):include
(Conference)..>(create):include
(Conference)..>(delete):include
(Conference)..>(getActiveConferenceByUser):include
(Conference)..>(findAllActiveByCurrentUser):include
(ConferenceUser)..>(getAllParticipation):include
(ConferenceUser)..>(getAllDeposit):include
(ConferenceUser)..>(saveOrCreate):include
```

Figure 10 demonstrates an example of the use case diagram for the ng-tracker project generated with the PlantUML system.

We plan to add the following improvements to the subsystem for generating use case diagrams:

- Use additional information from the knowledge base to improve the quality of the generated diagrams;
- Add extend and generalization relations support;
- Use natural language processing methods and linguistic environment W^B to generate more correct (in terms of UML notation) names for use cases.

Thus, the subsystem for use case diagram generation can automate the formation of the CS at the planning stages for customer requirements definition or at the design stage to consider the experience of previous projects.

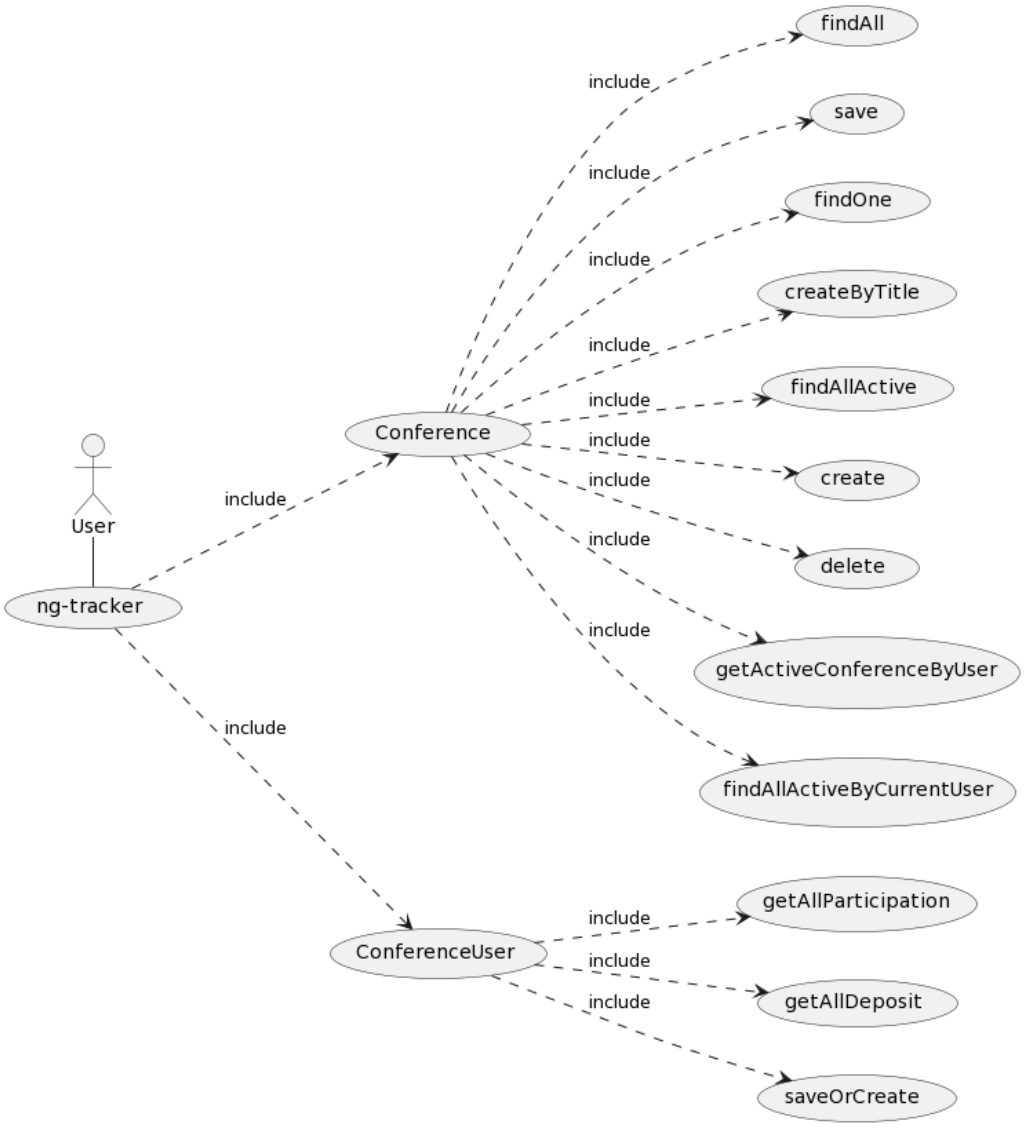

Figure 10. Example of the use case diagram for the ng-tracker project generated with the PlantUML system.

4.3. Diagnostic Analytics of Software Projects

For example, the tabbychat project [37] contains the experience of previous projects, and the ng-tracker project is currently being developed. We generated recommendations for the ng-tracker project based on data from the tabbychat project. We chose these repositories because the projects are written in Java and have comparable indicators.

We use the data of the following representations as initial data for the software project diagnostics:

- The representation of the development process L^B (Figure 3);
- The representation of domain features D^B (Figure 6).

Table 1 presents the time series extracted from these projects.

Table 1. Time series extracted from analyzed repositories.

Project	Indicator (Number)	Month				
		1	**2**	**3**	**4**	**5**
tabbychat	Commits	57	64	76	117	130
	Contributors	3	4	6	6	6
	Entities	7	7	7	7	7
	Processes	158	164	164	164	164
ng-tracker	Commits					251
	Contributors	1	1	8	8	8
	Entities	5	5	18	18	19
	Processes	115	129	206	271	273

We extracted the set of states from the analyzed repositories after applying the method for decision support in project management (Section 3.3). The set of states is presented in Table 2. We took the value ranges for the team size indicator from the development guidelines [38]. We approximated the value ranges for the number of commits indicator to a time interval of 1 month based on the paper [39]. We selected empirically the ranges for the number of entities and business processes indicators.

Table 2. States of indicators of analyzed repositories.

Project	Indicator (State)	Month				
		1	**2**	**3**	**4**	**5**
tabbychat	Commits	middle	middle	few	few	few
	Contributors	middle	middle	few	few	few
	Entities	few	few	few	few	few
	Processes	middle	middle	middle	middle	middle
ng-tracker	Commits	few	few	few	few	few
	Contributors	few	few	middle	middle	middle
	Entities	few	few	middle	middle	middle
	Processes	middle	middle	few	few	few

The invited expert proposed the following recommendations for the ng-tracker project based on the analysis of indicator states:

An increase in the number of entities with a decrease in the number of implemented business processes indicates the lack of progress in the development of new project functionality. Developers should create more business methods.

The expert made this conclusion, since the 'number of entities' indicator was stable in the tabbychat project, while this indicator increased in the ng-tracker project. Moreover, in the ng-tracker project, there is a decrease in the number of implemented business methods.

In the future, we plan to add a [40] decision support module to the diagnostic analytics subsystem, which allows for the automatation of generating recommendations based on expert knowledge.

5. Discussion

The proposed approach to design automation and project management makes it possible to formalize various features of existing software projects. The knowledge base formed in analyzing existing projects makes it possible to search, extract, and analyze design and management solutions that can be used by designers to form the CS in the design process and by project managers in the initial states of the project.

In Section 2, we analyzed various works that described software design automation. In most cases, the authors of these studies proposed models and methods for representing experience and knowledge to organize corporate knowledge bases by formalizing design artifacts of various types [3,9–14].

We also considered papers about the analysis of open-source software repositories to evaluate the quality of the repository, depending on various design, construction, and project management practices [16–21].

The main difference of the proposed approach from the existing ones is considering the various features of software projects:

- Development process features;
- Structure features;
- Environment features;
- Domain features;
- Project in dynamic representation.

Considering the dynamics of the project allows us to evaluate the impact of management decisions on the quality of design artifacts and the development process. Moreover, information about the dynamics of the project development can be used in predictive analytics methods to predict the occurrence of specific events.

Such a multimodal representation of the project allows us to find hidden patterns and dependencies between the various features of the project. We can use formalized features of various projects as a data set for data mining and machine learning methods.

We presented in the 'Results' section of this article some use cases of using the proposed approach.

The Information retrieval of projects allows us to more accurately find projects for research. The current implementation allows to search for projects, considering the domain features. For example, such projects can be used to research data models and/or business processes for a new (unknown) domain. In addition, the proposed information retrieval method allows us to consider the size of the team and the size of the project. This search options allows us to search only training or demonstration projects or only large projects. In the future, we plan to add a search for projects by dependencies and architectural solutions.

The method for generating use case diagrams allows us to assess the functionality of the project when making management decisions or choosing a project for research. In the future, we plan to add support for generating the following structural UML diagrams:

- A class diagram;
- A composite structure diagram;
- A component diagram;
- A deployment diagram;
- A package diagram.

The method for the diagnostic analytics of a project allows us to extract and compare the development trends of the current project with other successful or unsuccessful projects. In the future, we plan to automate the process of project analysis, extracting the project development trend, and generating recommendations to support decision-making.

We do not fully use all views of the knowledge base. For example, the representation of the software project structure P^B (Figure 4) can be used to find projects with a similar structural organization. Some researchers suggest that the way developers organized a project affects its success and code maintainability [41].

The disadvantages of the proposed approach are as follows:

- The need for expertise to adapt the approach to different programming languages and technologies;
- The need for expertise to consider the features of the development process;
- The need to use the project-hosting API (GitHub, GitLab) to extract information about the development process: stages, tasks, merge requests, etc.

We plan to add the following features:

- Support for fuzzy logic;
- Generating new types of design artifacts;
- The automatic generation of project management recommendations;
- Data mining methods.

6. Conclusions

This article discusses an approach to the design automation and project management of software projects. Design automation improves the quality of software projects by considering successful and unsuccessful design decisions based on the experience of previous projects.

We proposed the knowledge base model to solve the problem of design automation. The knowledge base allows for the formalization of various design artifacts and various indicators of the development process. The generated knowledge base can also be used as a source of a set of time series.

We proposed the diagnostic analytics method to support the project development process. The proposed method is based on the analysis of multiple time series to form recommendations for improving the quality and efficiency of project management decisions.

Author Contributions: Conceptualization, A.F. and A.R.; methodology, A.F. and A.R.; software, A.F., A.R., A.S. and J.S.; validation, A.F., A.R., A.S. and J.S.; formal analysis, A.F. and A.R.; investigation, A.F. and A.R.; resources, A.S. and J.S.; data curation, A.F. and A.R.; writing—original draft preparation, A.F. and A.R.; writing—review and editing, N.Y.; supervision, N.Y.; project administration, N.Y.; funding acquisition, N.Y. All authors have read and agreed to the published version of the manuscript

Funding: This research was funded by the Ministry of Science and Higher Education of the Russian Federation in the framework of the state task no.075-00233-20-05 "Research of intelligent predictive multimodal analysis of big data, and the extraction of knowledge from different sources".

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AD	Architectural Description
OS	Operational Space of Design Activity
CS	Conceptual Space of Design Activity
DT	Design Thinking Methodology
DL	Description Logic
HTTP	HyperText Transfer Protocol
REST	Representational State Transfer
API	Application Programming Interface
SHA	Secure Hash Algorithms
MVC	Model–view–controller
UML	Unified Modeling Language

References

1. Chaos Reports 1994–2017. Available online: http://www.standishgroup.com (accessed on 2 February 2023).
2. Ralph, P. The sensemaking-coevolution-implementation theory of software design. *Sci. Comput. Program.* **2015**, *101*, 21–41.
3. Sosnin, P. Substantially evolutionary theorizing in designing software-intensive systems. *Information* **2018**, *9*, 91.
4. Ferreira Martins, H.; Carvalho, de Oliveira, A., Jr.; Dias, Canedo, E.; Dias, Kosloski, R.A.; Ávila, Paldês, R.; Ávila Paldês, R.; Costa Oliveira, E. Design thinking: Challenges for software requirements elicitation. *Information* **2019**, *10*, 371.

5. Hoda, R.; Murugesan, L.K. Multi-level agile project management challenges: A self-organizing team perspective. *J. Syst. Softw.* **2016**, *117*, 245–257.
6. da Silva F.Q.; Costa C.; Franca A.C.C.; Prikladinicki R. Challenges and solutions in distributed software development project management: A systematic literature review. In Proceedings of the 5th IEEE International Conference on Global Software Engineering, Princeton, NJ, USA, 23–26 August 2010; pp. 87–96.
7. Niazi, M.; Mahmood, S.; Alshayeb, M.; Riaz, M.R.; Faisal, K.; Cerpa, N.; Khan, S.U.; Richardson, I. Challenges of project management in global software development: A client-vendor analysis. *Inf. Softw. Technol.* **2016**, *80*, 1–19.
8. Engwall, M.; Jerbrant, A. The resource allocation syndrome: The prime challenge of multi-project management? *Int. J. Proj. Manag.* **2003**, *21*, 403–409.
9. Bedjeti, A.; Lago, P.; Lewis, G.A.; De Boer, R.D.; Hilliard, R. Modeling context with an architecture viewpoint. In Proceedings of the IEEE International Conference on Software Architecture (ICSA-2017), Gothenburg, Sweden, 3–7 April 2017; pp. 117–120.
10. Wongthongtham, P.; Pakdeetrakulwong, U.; Marzooq, S.H. Ontology annotation for software engineering project management in multisite distributed software development environments. In *Software Project Management for Distributed Computing*; Mahmood, Z., Eds.; Springer: Cham, Switzerland, 2017; pp. 315–343.
11. Namestnikov, A.; Guskov, G. Ontological mapping for conceptual models of software system. In Proceedings of the Open Semantic Technologies for Intelligent Systems Conference, Minsk, Republic of Belarus, 16–18 February 2017; pp. 16–18.
12. Guskov, G.; Namestnikov, A.; Yarushkina, N. Approach to the search for similar software projects based on the UML ontology. In Proceedings of the International Conference on Intelligent Information Technologies for Industry, Varna, Bulgaria, 14–16 September 2017; pp. 3–10.
13. Bechberger, L.; Kühnberger, K.U. A thorough formalization of conceptual spaces. In Proceedings of the Joint German/Austrian Conference on Artificial Intelligence, Dortmund, Germany, 25–29 September 2017; pp. 58–71.
14. Di Noia, T.; Mongiello, M.; Nocera, F.; Straccia, U. A fuzzy ontology-based approach for tool-supported decision making in architectural design. *Knowl. Inf. Syst.* **2019**, *58*, 83–112.
15. ISO/IEC/IEEE 42010:2022. Software, systems and enterprise – Architecture description. Available online: https://www.iso.org/standard/74393.html (accessed on 27 November 2022).
16. Borle, N.C.; Feghhi, M.; Stroulia, E.; Greiner, R.; Hindle, A. Analyzing the effects of test driven development in GitHub. *Empir. Softw. Eng.* **2018**, *23*, 1931–1958.
17. Henkel, J.; Bird, C.; Lahiri, S.K.; Reps, T. Learning from, understanding, and supporting devops artifacts for docker. In Proceedings of the 42nd International Conference on Software Engineering (ICSE-2020), Seoul, Korea (South), 5–11 October 2020; pp. 38–49.
18. Schermann, G.; Zumberi, S.; Cito, J. Structured information on state and evolution of dockerfiles on GitHub. In Proceedings of the 15th international conference on mining software repositories, Gothenburg, Sweden, 28–29 May 2018; pp. 26–29.
19. Xia, T.; Fu, W.; Shu, R.; Agrawal, R.; Menzies, T. Predicting health indicators for open source projects (using hyperparameter optimization). *Empir. Softw. Eng.* **2022**, *27*, 1–31.
20. De Stefano, M.; Pecorelli, F.; Tamburri, D.A.; Palomba, F.; De Lucia, A. Splicing community patterns and smells: A preliminary study. In Proceedings of the 42nd international conference on software engineering workshops, Seoul, Korea (South), 27 June–19 July 2020; pp. 703–710.
21. Tamburri, D.A.; Palomba, F.; Serebrenik, A.; Zaidman, A. Discovering community patterns in open-source: A systematic approach and its evaluation. *Empir. Softw. Eng.* **2019**, *24*, 1369–1417.
22. Bhatia, M.P.S.; Kumar, A.; Beniwal, R. Ontologies for software engineering: Past, present and future. *Indian J. Sci. Technol.* **2016**, *9*, 1–16.
23. Isotani, S.; Bittencourt, I.I.; Barbosa, E.F.; Dermeval, D.; Paiva, R.O.A. Ontology driven software engineering: A review of challenges and opportunities. *IEEE Lat. Am. Trans.* **2015**, *13*, 863–869.
24. Rudolph, S. Foundations of description logics. In Proceedings of the 7th International Summer School on Reasoning Web. Semantic Technologies for the Web of Data, Galway, Ireland, 23–27 August 2011; pp. 76–136.
25. OWL 2 Web Ontology Language. Structural Specification and Functional-Style Syntax (Second Edition). Available online: https://www.w3.org/TR/owl2-syntax/ (accessed on 2 February 2023).
26. ng-tracker repository on GitLab. Available online: https://gitlab.com/romanov73/ng-tracker (accessed on 2 February 2023).
27. Filippov, A.; Romanov, A.; Iastrebov, D. An Approach to Data Mining of Software Repositories in Terms of Quantitative Indicators of the Development Process and Domain Features. In Proceedings of the International Conference on Intelligent Information Technologies for Industry, Istanbul, Turkey, 31 October–6 November 2022; pp. 346–357.
28. GitLab REST API Documentation. Available online: https://docs.gitlab.com/ee/api/rest/ (accessed on 2 February 2023).
29. Eclipse JGit Official Website. Available online: https://www.eclipse.org/jgit/ (accessed on 2 February 2023).
30. Pecherskikh, A.A.; Romanov, A.A.; Beresnev, I.I. An approach for searching software system projects with similar structure. *Autom. Control. Process.* **2022**, *3*, 20–26.
31. Namestnikov, A.M.; Filippov, A.A.; Avvakumova, V.S. An ontology based model of technical documentation fuzzy structuring. In Proceedings of the 2nd International Workshop on Soft Computing Applications and Knowledge Discovery (SCAKD 2016), Moscow, Russia, 18 July 2016; pp. 63–74.

32. Yarushkina, N.; Filippov, A.; Grigoricheva, M.; Moshkin, V. The Method for Improving the Quality of Information Retrieval Based on Linguistic Analysis of Search Query. In Proceedings of International Conference on Artificial Intelligence and Soft Computing (ICAISC-2019), Zakopane, Poland, 16–20 June 2019; pp. 474–485.
33. Romanov, A.A.; Filippov, A.A.; Voronina, V.V.; Guskov, G.Y.; Yarushkina, N.G. Modeling the Context of the Problem Domain of Time Series with Type-2 Fuzzy Sets. *Mathematics* **2021**, *9*, 2947.
34. Rahman, M.M.; Chakraborty, S.; Kaiser, G.; Ray, B. A case study on the impact of similarity measure on information retrieval based software engineering tasks. *arXiv* **2018**, arXiv:1808.02911. Available online: https://arxiv.org/pdf/1808.02911.pdf (accessed on 27 November 2022).
35. Holzschuher, F.; Peinl, R. Performance of graph query languages: Comparison of Cypher, Gremlin and native access in Neo4j. In Proceedings of the Joint EDBT/ICDT 2013 Workshops, Genoa, Italy, 18–22 March 2013; pp. 95–204.
36. PlantUML. UML Diagram Generator. Available online: https://plantuml.com (accessed on 27 November 2022).
37. Tabbychat. Plugin for Minecraft. Available online: https://github.com/killjoy1221/tabbychat (accessed on 2 February 2023).
38. The 2020 Scrum Guide: Scrum Team. Available online: https://www.scrumguides.org/scrum-guide.html#scrum-team (accessed on 27 November 2022).
39. Grönlund, M.; Jefford-Baker, J. *Measuring Correlation between Commit Frequency and Popularity on GitHub*; School of Computer Science and Communication (CSC): Stockholm, Sweden, 2017.
40. Romanov, A.; Yarushkina, N.; Filippov, A. Application of time series analysis and forecasting methods for enterprise decision-management. In Proceedings of the International Conference on Artificial Intelligence and Soft Computing, Zakopane, Poland, 12–14 October 2020; pp. 326–337.
41. Zhu, J.; Zhou, M.; Mockus, A. Patterns of folder use and project popularity: A case study of GitHub repositories. In Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, Torino, Italy, 18–19 September 2014; pp. 1–4.