UDC 004.415.2. (043.3) https://doi.org/10.15407/pp2020.04.110

N. Sydorov

TOWARD SOFTWARE ARTIFACTS ECOSYSTEM

In the process of developing and maintaining a software product, many things are created and used that are called software artefacts. Software artifacts are changed, reused, and change relationships in the development and maintenance processes of a software product. The complexity and variety of software artifact relationships require adequate means of description and management. They may be a software artifacts ecosystem. In the article, for the first time, a concept of a software artifact ecosystem is proposed. The concept describes a generic model of the software artifacts ecosystem, which is the Cornerstone ecosystem type and consists of three actors – the platform, the software, and the artifact. Based on the generic model, the SD model of the software artifacts ecosystem is described. The roles of actors in the ecosystem are indicated, the relationships between actors are described. The developer's activities will be more efficient, the software is understandable, and the development and maintenance is cheaper when the styles (standards) are used. As case study, based on the generic model of the software artifacts ecosystem, a declarative model of the programming style ecosystem has been developed. Three-level model of programming style artifact is proposed. The tools and processes for creating and using a programming style artifact are developed and described.

Key words: software engineering, software artifact, software ecosystem, programming, programming style, ontology.

1. Introduction

In the processes of developing and maintaining software products, many things are created and used, which are called artifacts. Artifacts can be different in form and presentation. They can be part of a software product or provide processes for its development and maintenance, be intermediate results of processes, or be part of other artifacts. Thus, there is a huge variety of software artifacts, including design plans, work products (specifications, architectural and detailed designs, code, and documentation), user stories, bug reports, tools, including for processing artifacts, but not limited to this. Various and often complex connections are established between artifacts. Artifacts change, reuse, and change links in the development and maintenance of a software product. Therefore, artifacts play an important role in the software life cycle whatever of its model and require the attention of all interested parties.

The software industry is constantly evolving and changing. Not only products and technologies are developing. Many software companies are experimenting with new business models, leading to fundamental changes in the structures of both the company and its client. Recently, many companies have been using the concept of “software ecosystem” to describe development, creating them around themselves or their products, taking into account customer connections. Ecosystems have shown themselves to be a promising management tool, an evolving software product.

The complexity and diversity of software artifact relationships require adequate description and management tools. This could be a software artifacts ecosystem. Such an ecosystem points to more detailed level than a software ecosystem, but at this level most of the approaches, methods and tools that are used in a software ecosystem can be used.

In the article, for the first time, a model of a software artifacts ecosystem is proposed. Its application is shown on the case of a programming style ecosystem. Within the conception framework, a generalized model of the software artifacts ecosystem is described. The ecosystem belongs to the Cornstoun ecosystem type and consists of three actors – platform, software and artifact. The roles of actors in the ecosystem are indicated, connections between actors are described. The types, rules, and attributes of actors, relationships, and actions can be refined for specific software artifacts ecosystem models. The same applies to analyzing ecosystem properties.

Based on the generic model of the software artifacts ecosystem, a declarative model of the programming style ecosystem has been developed. The programming style

is an artifact that plays an important role in the development and maintenance of software. The description of the processes of creation and the use of the programming style is made by using the ontology.

2. Related works

***Software artifacts*.** In the software life cycle to support the processes of creating and maintaining a software product, many different artifacts are created and used. Wide ranges of components are considered as artifacts, from documentation, work products and their parts, to auxiliary tools. Interacting, artifacts ensure the efficient execution of software life cycle processes.

In [1], artifacts are analyzed in the context of reuse as equipment in the sense of work [2]. At the same time, three goals (writing, processing and transferring artifacts) and three aspects of equipment (the in-order-to of equipment, readiness-to-hand, presence-and-hand) are considered. In addition, since artifacts are analyzed as reusable components that are embedded in the created software product, their characteristics are taken into account: holism, commonality, reusability and maturity. Considering an artifact as hardware – a thing built into the context of a software product, the interaction of the specified characteristics of software artifacts is investigated. In [3], artifacts are considered in the context of a software product line and are divided into three types – architecture, shared components, and components made from shared ones. For each type of artifacts, three levels of maturity are identified, depending on the degree of integration of the artifact of the corresponding type into the software product line. In [4], artifacts are considered as information parts that are created, modified and used in the RUP processes. Artifacts can be of different types and take different forms, from UML models to executable code, and can be used in the creation and maintenance of a software product. Artifacts are the input and output of actions in RUP processes. In [5], software documentation as an artifact is considered. Artifact as a means of representing information about software is defined. A maintenance model of documentation as a software artifact is introduced.

Artifact modeling. In the following works, attempts are made to build a model of the artifact.

The paper [6] presents a metamodel for software artifacts aiming at providing a new and structured way to represent artifact content, other than current sections hierarchy. This work defines an extension to UML/MOF and SPEM meta-models by means of layers. The paper [7] discusses the theoretical foundations for the representation and interpretation of software artifacts. Based on different levels of perception of artifacts by a person – the user of artifacts introduces three levels of representation of artifacts – physical (physical representation), structural (syntactic structure) and semantic (semantic content). In addition, two steps for processing artifacts - parsing the physical representation, and analyzing the syntactic structure – the result of the first step (interpretation) are introduced. A meta model of artifacts is built on the basis of presentation levels and processing steps. The work [8] considers the architecture of tools that provide the creation and maintenance of metadata about software artifacts, which form an environment consisting of resources – development artifacts. Tools to manage the artifact environment are used.

***Artifacts in software development*.** The experience of using artifacts in life cycle processes in several works is considered.

In [9], artifact-oriented development of embedded systems is considered. A conceptual model of artifact-oriented development is proposed, examples of its use are given. In [10], artifact-oriented model-driven development is considered. Details a better understanding on how explicating artifacts and their relations facilitates traceability of artifacts, change impact analysis, and interoperability of software tools are considered. The paper [11] concentrates on the paradigm artefact-orientation in requirements engineering and presents a meta model. This meta model is inferred from two concrete domain specific requirements engineering models: one for the application domain of embedded systems and one for the application domain of business information systems. In [12] shown, that ccollaborative development of software products across organizational boundaries in software

ecosystems adds new challenges to existing software engineering processes. A new approach offered for handling the diverse software artefacts in ecosystems by adapting features from social network sites. In paper [13], an industrial survey to create an Activity-Based Artifact Quality Model to define what this means from a stakeholder's viewpoint is proposed. Specifically was conducted. Quality factors of test artifacts that have a positive or negative impact on the activities of Agile testers are explored. Quality model contains 16 quality factors for six test artifacts that are reportedly relevant to at least five stakeholders in the process. In paper [14] reference model and a metamodel for traceability are proposed. The reference model, defined by the conceptual basis, may be used in the creation of traceability approaches. The reference model was used to develop a metamodel. In paper [15], a generic artifact model based on an empirical investigation is proposed. The results of a mapping study in combination with a systematic literature review to analyze the usage of artifacts in agile methods are presented.

Towards a software artifacts ecosystem. We are not aware of any work directly devoted to the consideration of problems associated with the study of software artifacts ecosystems. However, there are works, the results of which can be used to solve these problems. In [16], attention is rightly drawn to the fact that in software ecosystems, attention is now paid to the participants only at the top level - these are organizations and teams that create, implement and maintain software products. However, there is a lower level – artifacts, the role of which in the life cycle processes can hardly be overestimated. In [17], there are requirements for describing and analyzing software ecosystems, which in our paper to model software artifacts ecosystems are used.

3. The generic model of software artefacts ecosystem

This section discusses a generic model of the software artifact ecosystem. Several methods are now used to model software ecosystems [18]. The application of a particular method depends on the type of ecosystem and the goals of the modeling. To represent the software artifacts ecosystem, this work uses the i* modeling approach [19]. In contrast to the most commonly used SSN method, which focuses on describing the software ecosystem at the top level (product, developer, vendor, user), the i* approach provides a description of the ecosystem of a more detailed software presentation layer that corresponds to the level software artifacts. Fig. 1 presents generic model of the software artifacts ecosystem. When designing an ecosystem, two groups of requirements are used [17]: descriptive and analytical.

The first group includes the requirements for the definition of actors, connections between them and their actions. In addition, the requirements for determining the types, rules and attributes of actors, connections and actions are formulated, as well as the requirements for determining the specific characteristics of both the ecosystem as a whole and its elements, for example, productivity, efficiency, security.

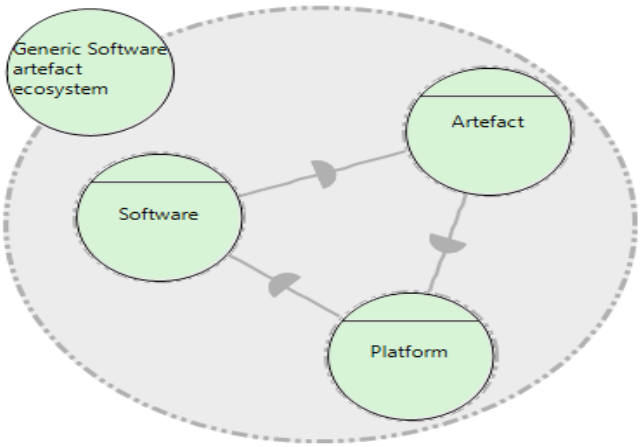

Fig. 1. The generic model of the software artifacts ecosystem

The second group includes requirements for defining characteristics that provide analysis of the ecosystem from incentives and motivation to sustainability and productivity.

Table the actors and roles in the software artifacts ecosystem are given (Tabl 1). The software artifacts ecosystem belongs to the Cornerstone type, since the basis of the ecosystem is a technological platform for the development and maintenance of software, the functionality of which is extended by using artifacts [20]. Thus, the actors of the ecosystem are a platform with a management role, software with a software product role, an artifact with a support service provider role. Common connections between actors can be indicated (Fig. 2). The platform, in the context of which such components as the life cycle model, organizational and technical support for development and maintenance are considered, defines and uses the artifact as an auxiliary means of implementing processes and filling the structure of a software product. The software depends on the platform, which is the main mean for the implementation of development and maintenance processes. The platform uses the artifact directly as a component in the software structure or indirectly as a means of improving the efficiency of the platform's processes.

The types, rules, and attributes of actors, relationships, and actions can be refined for specific ecosystem models of a software artifact. The same applies to meeting the requirements for the analysis of ecosystem properties [17].

Table 1. Actors and roles in the software artifacts ecosystem

Ecosystem type	**Actors**	**The role of the actor in the ecosystem**
Cornstoun ecosystem	Platform	Orchestration
	Software	Product
	Artefact	Support service provider

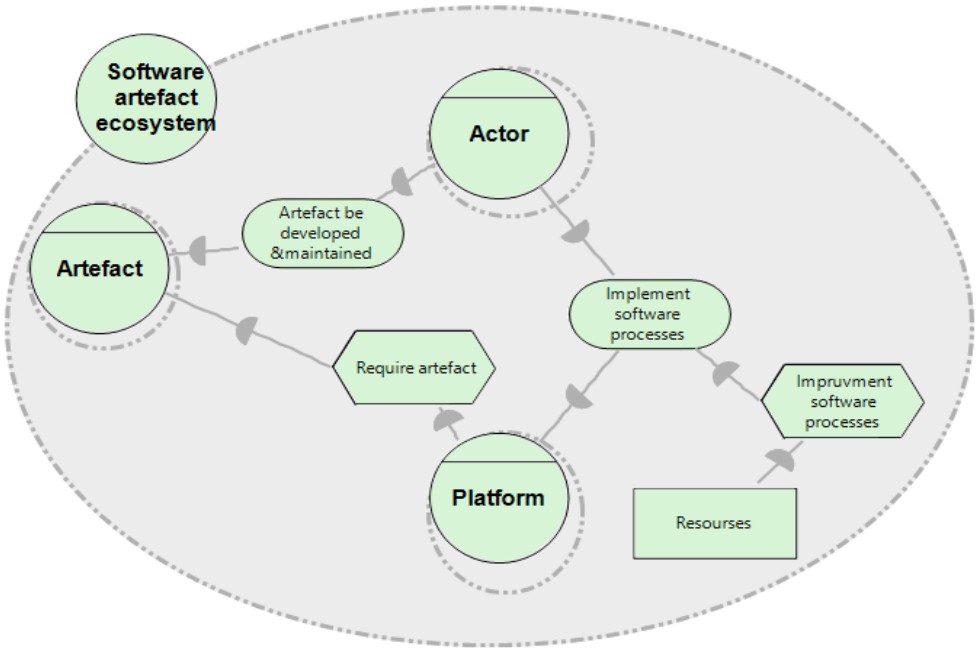

Fig. 2. The SD model of the software artefact ecosystem

4. Case study. The programming style ecosystem

Today, methods and tools that are based on reuse have become widespread for the development and maintenance of software products. The application of these methods and tools requires the developer to read, analyze and understand a significant number of representations of work products from different phases of the software life cycle. Reuse is now widespread from requirements specifications to source code and documentation. Therefore, one of the main requirements for software is understandability. The developer's activities will be more efficient, the software is understandable, and the development and maintenance is cheaper when the styles (standards) are use. They will ensure that the work products of different phases of the life cycle are understandable [21].

Fig. 3 shows the model of a programming style ecosystem, which is built based on a generic model of a software artifact ecosystem (Fig. 1).

The artifact in this model is the programming style, and the actor, the software, is represented by that part of it – the source code for which the programming style is applied. Artifact – the programming style is platform-specific, as the style rules depend on a number of platform conditions, such as the programming language, management goals, schedule, risks, and project budget. The programming style is used in the source code construction (the phase of software live cycle) and affects the efficiency of the construction and maintenance processes.

Based on the artefact model from work [7], described the programming style artefact by the three levels of perception and the two processing steps (Fig. 4).

Level 1 – Semantic content. The content represents the meaning of an artefact. The content is interpreted in the context of the individual knowledge of the stakeholder (programmer) or the interpreter of the machine (in this case – Protégé The content is based on the rules of the programming style, which are described by the ontology.

Level 2 – Syntactic Structure. The structure of an artifact represents the syntactic expression of its content. The structure of the artifact is described in Web Ontology Language (OWL).

Level 3 – Physical Representation. The artefact is represented in the file of OWL text format.

There are two the processing steps.

Processing Step 1 – Parsing. The outcome of the parsing process is the syntax structure of the artefact. This process is performed by the OWL parser implemented using the OWL API [22].

Processing Step 2 – Interpretation. Interpretation is the process of extracting the content (i.e. the meaning) from the structure. This process is performed by Protégé system.

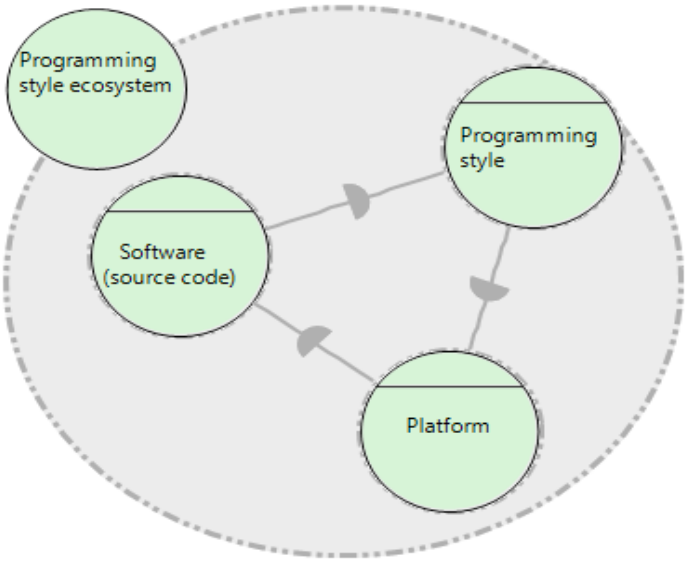

Fig. 3. The SD model of programming style ecosystem

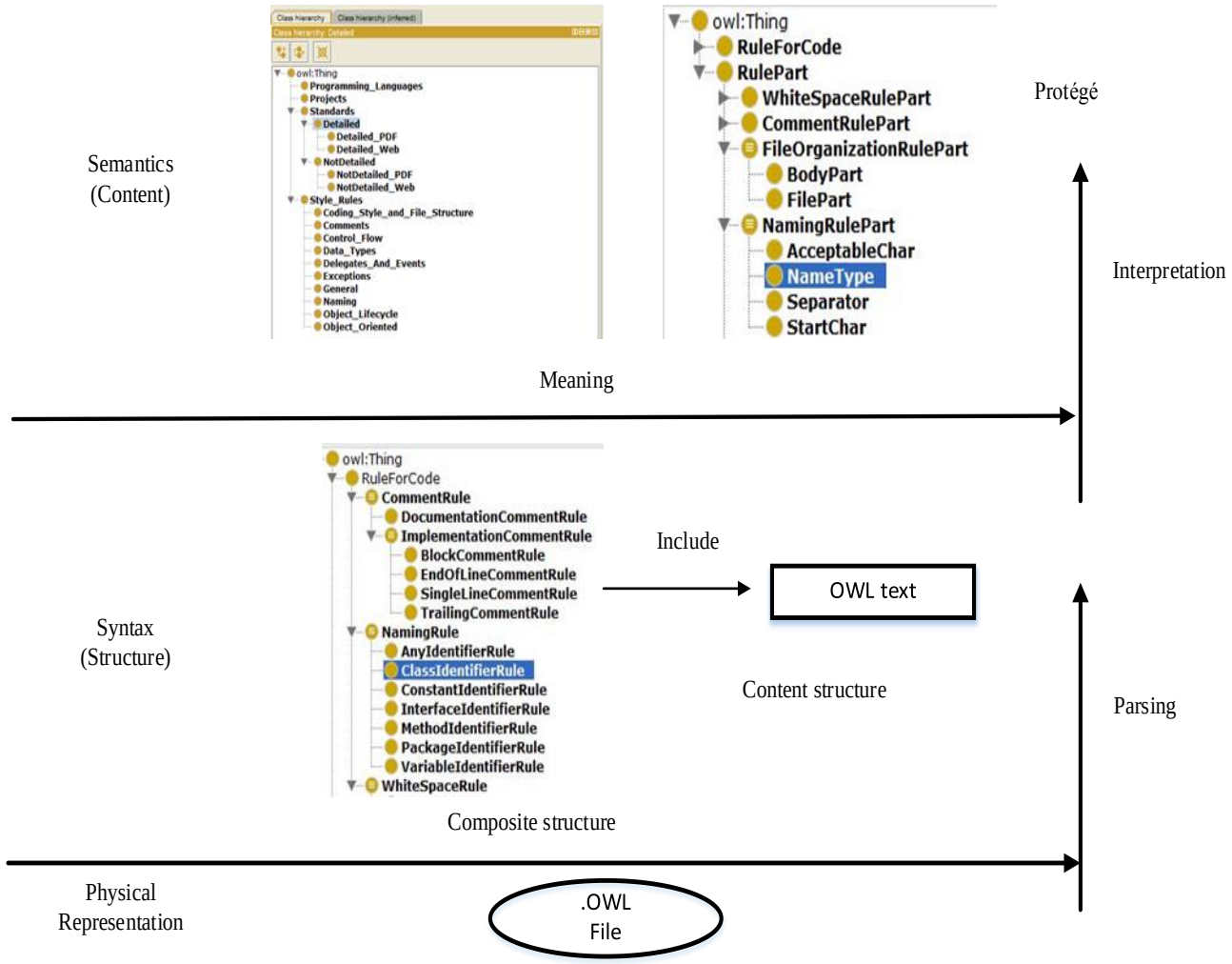

Fig. 4. The levels of perception of programming style artefact

The characteristics of the domain in which the style is applied are given in tab. 2. [21]. The activities of a programmer in a domain are shown in Fig. 5. The use of the programming style as an artifact involves the implementation of three processes [23]: the creation of an artifact, as a result of which the programming language style is built, the use of the style when programs are writing and the process changed of the artifact.

Table 2. Characteristics of the style of the program domain

Epoch	**Characteristic**						
	Property			**Factors**			**Means**
	Period	**Idea**	**Principle of importance**	**Historical**	**Social**	**Style**	**Elements**
Application of standards in coding	2000	Readability, quality, safety	Productivity, maintenance	Standardization	Programming team, requirements	Modular, mega programming	Composition, classification in programming languages

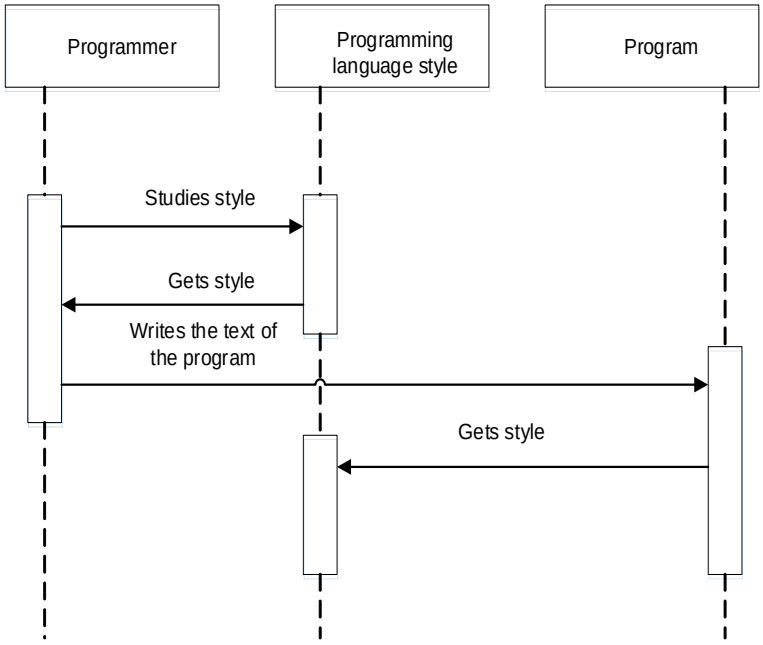

Fig. 5. Domain sequence diagram

In fig. 6. The ontology of creating a programming style is presented. The ontology describes in detail the participants and actions taking place in this regard in the programming style ecosystem. All ontology concepts are categorized as resources in i* terminology, with the exception of the <<event>> concept, which represents a goal. At the same time, the concepts Coding phase, Party, Programming language refer to the Platform actor, and the concepts Creating work product style, Style party create guide, Style and Programming language style to the Programming style actor.

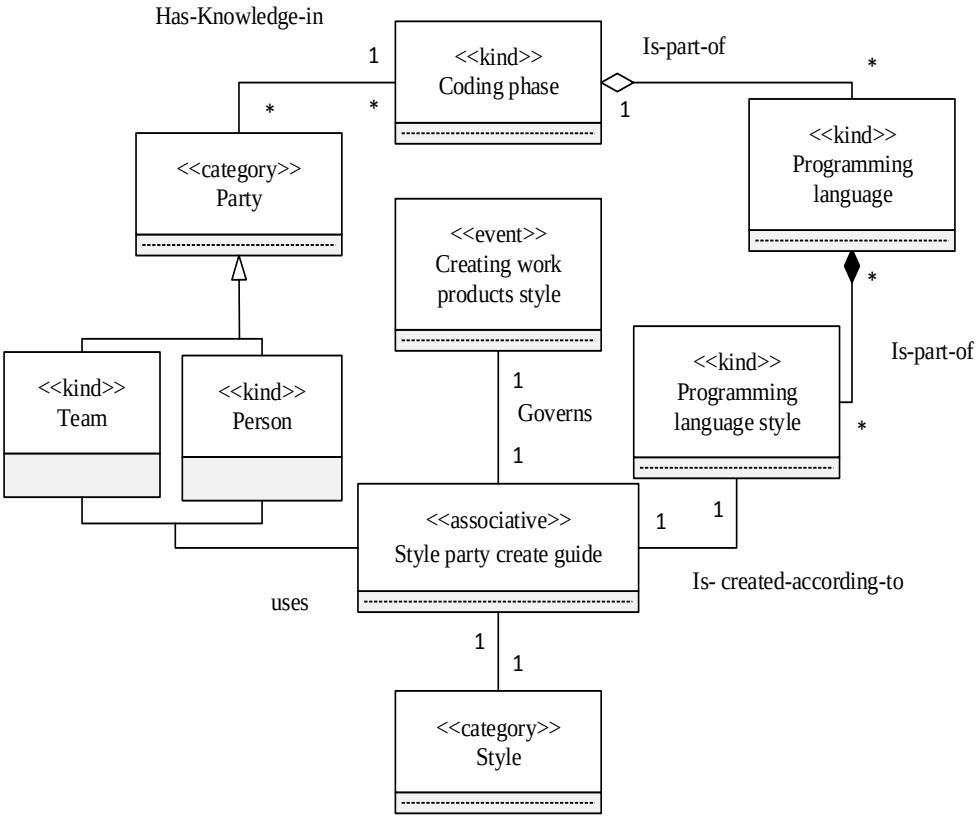

Fig. 6. Ontology of programming style creation

Fig. 7. The ontology of using the programming style is presented. An ontology describes the relevant actors and activities in a programming style ecosystem. The Party, Coding phase concepts belong to the Platform actor, the Program, Program style concepts to the Software actor, and the Using work product style, Style party using guide, Program language style concepts to the Programming style actor.

To implement the processes of creating and using a programming style, tools are created that can be considered, on the one hand, as resources of the Programming style artifact, and on the other hand, as artifacts as part of the Platform artifact. These include the programming style knowledge base and the Reasoner. Thus, the programmer, while coding the program, applies the ontology of the programming style, both for learning the style and for checking the observance of the style in the program. Therefore, two tools are needed - one to create an ontology and support the programmer in the coding process, and the second, to control the application of the programming style in the source code of the program (Fig. 8) [23].

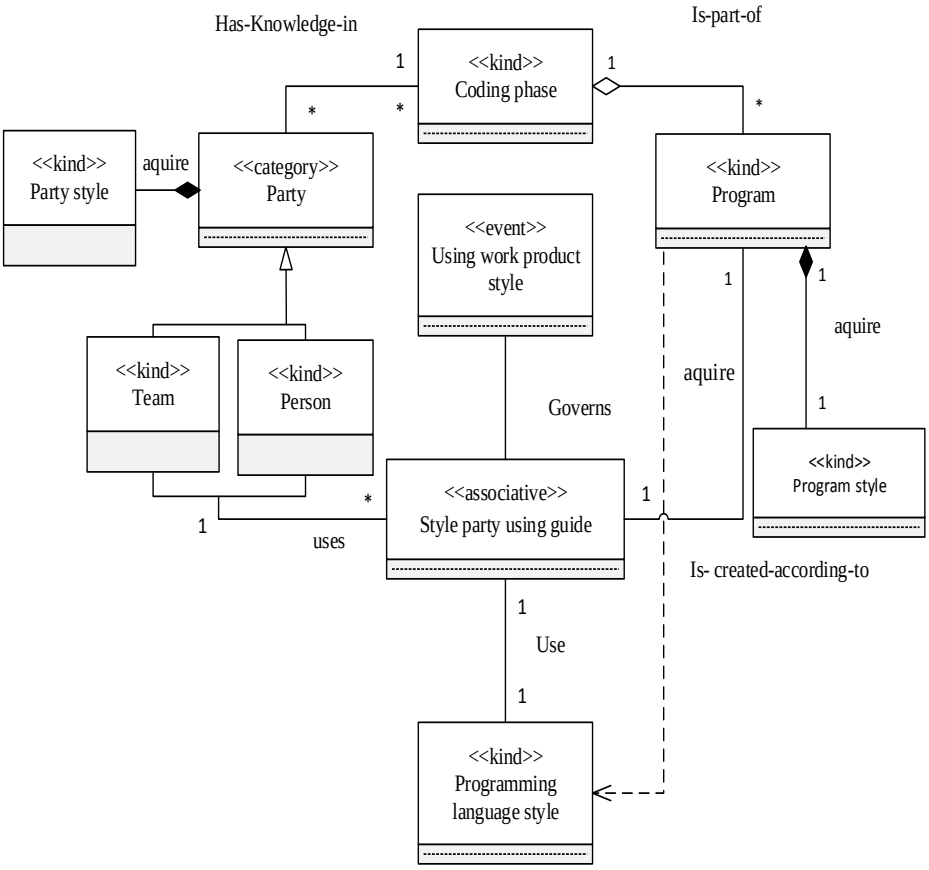

Fig. 7. Ontology of using programming style

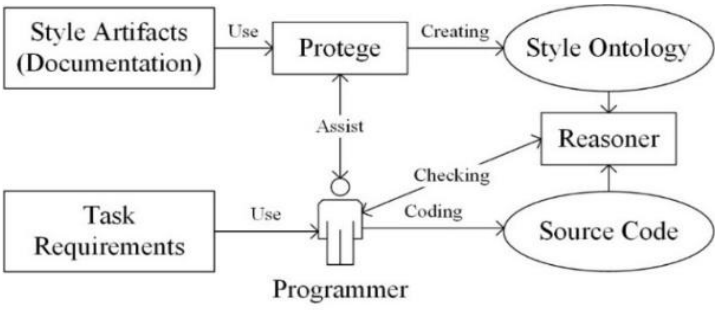

Fig. 8. Tools usage diagram

The style analyst, using the first tool - Protégé, setting up the ontology to the appropriate programming style, creating a TBox (Fig. 6). After setting up, the programmer is introduced to the programming style with the help of Protégé. The second tool is functionally similar to the reasoner, but adds a function for identifying style errors. In terms of descriptive logic, the reasoner verifies the consistency of the ontology (Fig. 9).

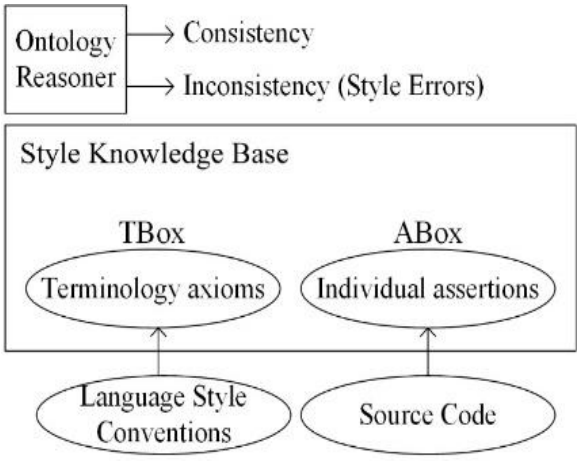

Fig. 9. Knowledge base of programming style

Protégé is used to create TBox. It is part of an ontology with terms describing a programming style. Assertions about the source code (ABox) written by the programmer are created by the corresponding part of the reasoner. It provides the appropriate service using the knowledge base (TBox and ABox). The service includes, firstly, the verification of the consistency of the ontology (a direct function of the reasoner), and secondly, the search for stylistic errors in the source code of the program.

5. Results Analysis and Discussion

The results are a development of the solutions obtained in the works of the author [21–24]. For the first time, the concept of the software artifact ecosystem is proposed. Within the framework of the concept, the generic model of the software artifact ecosystem is described. Model belongs to the Cornstoun ecosystem type and consists of three actors – platform, software, and artifact. The roles of actors in the ecosystem are indicated, connections between actors are described. Based on the generic model of the software artifact ecosystem, a declarative model of the programming style ecosystem has been developed. The programming style is an artifact that plays an important role in the development and maintenance of software. Using [7], a three-level artifact model of - programming style is proposed. The description of the processes of creating and using a programming style is made by applying the ontology. In continuation of research of the software artifact ecosystem, the description of the actors of the ecosystem will be expanded and the types, rules and attributes of actors, links and actions will be developed. In addition, the metric provision of the ecosystem in relation to determining the effectiveness, sustainability and reliability of the ecosystem of software artifacts will consider.

The work done in research “Research on software artifacts ecosystems”, № 0120U104329.

References

1. Nuwangi S.M., Darshana S. Software artefacts as equipment: a new conception to software development using reusable software artefacts. Thirty-Sixth International Conference on Information Systems. 2015. Texas, USA.
2. Heidegger M. (1927/1962) Being and Time, Translated by John Macquarrie & Edward Robinson. USA: Harper & Row.
3. Bosch J. Maturity and Evolution in Software Product Lines: Approaches, Artefacts and Organization, Software Product Lines, Second International Conference, SPLC 2, San Diego, CA, USA, August 19–22, 2002,
4. Rational Unified Process: Best Practices for Software development Teams, Rational Software White Paper TP026B, Rev. 11/01. 1998. 18 p.
5. Glass R. Software maintenance documentation, SIGDOC '89, Pittsburg, Pennsylvania, USA, ACM Press. 1989. P. 18 – 23.
6. Silva M., Oliveira T., Bastos R., Software Artifact Metamodel, XXIII Brazilian Symposium on Software Engineering, 2009. P. 176 – 186.
7. Fernandez D M., Bohm W., Broy M. Artefacts in Software Engineering: A Fundamental Positioning, International *Journal on Software and Systems Modeling*. 2018. 26. 9 p.

8. Dewar R.G. Managing Software Engineering Artefact Metadata, Department of Computer Science, Heriot-Watt University, Edinburgh, UK. (2005)
9. Bohm W., Vogelsang A. An Artifact-oriented Framework for the Seamless Development of Embedded Systems, *Model-Based Engineering of Embedded Systems*. Springer Berlin Heidelberg. 2012. P. 225–234.
10. Butting, A., Greifenberg T, Rumpe B. Wortmann: A. On the Need for Artifact Models in Model-Driven Systems Engineering Projects. In: *Software Technologies: Applications and Foundations*, LNCS 10748. Springer. 2018. P. 146–153.
11. Fernández D.M., Penzenstadler B., Kuhrmann M., Broy M., A Meta Model for Artefact-Orientation:Fundamentals and Lessons Learned in Requirements Engineering, Lecture Notes in Computer Science. October 2010.
12. Seichter D., Dhungana D., Pleuss A., Hauptmann B. Knowledge Management in Software Ecosystems: Software Artefacts as First-class Citizens. ECSA 2010. August 23–26, 2010. Copenhagen. Denmark. P. 119–126.
13. Fischbach J., Mendez D. What Makes Agile Test Artifacts Useful? An Activity-Based Quality Model from a Practitioners' Perspective, ESEM '20, October 8–9, 2020, Bari, Italy.
14. Azevedo B., Jino M., Modeling Traceability in Software Development: A Metamodel and a Reference Model for Traceability, ENASE, School of Electrical and Computer Engineering. University of Campinas, Brazil, 8 p.
15. Kuhrmann M., Fernández D., Towards Artifact Models as Process Interfaces in Distributed Software Projects, IEEE workshop proceedings, 10 p.
16. Seichter D., Dhungana D., Pleuss A., Hauptmann B. Knowledge Management in Software Ecosystems: Software Artefacts as First-class Citizens, ECSA 2010 August 23–26, 2010. Copenhagen. Denmark. P. 119–126.
17. Sadi M., Yu E. Designing Software Ecosystems: How Can Modeling Techniques Help? Springer-Verlag, Berlin Heidelberg. 2015. 15 p.
18. Sydorov N. Software Ecology. *Software Engineering*. 2010. P. 53–61.
19. Yu E. Modelling Strategic Relationships for Business Process Reengineering. Ph.D., thesis. Dept. of Computer Science, University of Toronto. 1995.
20. Knodel J., Manikas K. Towards a typification of software ecosystems. In Fernandes et al. Software Business – 6th International Conference. ICSOB 2015. Braga, Portugal. June 10–12, 2015. Proceedings 2015. vol. 210 of Lecture Notes in Business Information Processing. Springer. P. 60–65.
21. Sydorov N.A. Software Stylistics. *Problems of Programming*. 2005. 2,3. P. 245–254.
22. Sidorov N., Sidorova N., Pirog A. Ontology-driven tool for utilizing programming styles. *Вісник НАУ*. 2017. Том 71. № 2. С. 84–93.
23. Sydorov N., Sydorova N., Sydorov E., Cholyshkina O., Batsurovska I. Development of the approach to using a style in software engineering. *Eastern-European Journal of Enterprise Technologies*. 2019. 4/2 (100). P. 41–51.
24. Sydorov N.A., Sydorova N.N., Sydorov E.N. Description model of programming style ecosystem. Problems in programming, special issue. Proceeding of the UkrProg'2020. N 2–3. P. 74–81.

Література

1. Nuwangi S.M., Darshana S. Software artefacts as equipment: a new conception to software development using reusable software artefacts. Thirty-Sixth International Conference on Information Systems. 2015. Texas, USA.
2. Heidegger M. (1927/1962) Being and Time, Translated by John Macquarrie & Edward Robinson. USA: Harper & Row.
3. Bosch J. Maturity and Evolution in Software Product Lines: Approaches, Artefacts and Organization, Software Product Lines, Second International Conference, SPLC 2, San Diego, CA, USA, August 19–22, 2002,
4. Rational Unified Process: Best Practices for Software development Teams, Rational Software White Paper TP026B, Rev. 11/01. 1998. 18 p.
5. Glass R. Software maintenance documentation, SIGDOC '89, Pittsburg, Pennsylvania, USA, ACM Press. 1989. P. 18 – 23.
6. Silva M., Oliveira T., Bastos R., Software Artifact Metamodel, XXIII Brazilian Symposium on Software Engineering, 2009. P.176–186
7. Fernandez D M., Bohm W., Broy M. Artefacts in Software Engineering: A Fundamental Positioning, International *Journal on Software and Systems Modeling*. 2018. 26. 9 p.
8. Dewar R.G. Managing Software Engineering Artefact Metadata, Department of Computer

Science, Heriot-Watt University, Edinburgh, UK. (2005)
9. Bohm W., Vogelsang A. An Artifact-oriented Framework for the Seamless Development of Embedded Systems, *Model-Based Engineering of Embedded Systems*. Springer Berlin Heidelberg. 2012. P. 225–234.
10. Butting, A., Greifenberg T, Rumpe B. Wortmann: A. On the Need for Artifact Models in Model-Driven Systems Engineering Projects. In: *Software Technologies: Applications and Foundations*, LNCS 10748. Springer. 2018. P.146–153.
11. Fernández D.M., Penzenstadler B., Kuhrmann M., Broy M., A Meta Model for Artefact-Orientation:Fundamentals and Lessons Learned in Requirements Engineering, Lecture Notes in Computer Science. October 2010.
12. Seichter D., Dhungana D., Pleuss A., Hauptmann B. Knowledge Management in Software Ecosystems: Software Artefacts as First-class Citizens. ECSA 2010. August 23–26, 2010. Copenhagen. Denmark. P.119–126.
13. Fischbach J., Mendez D. What Makes Agile Test Artifacts Useful? An Activity-Based Quality Model from a Practitioners' Perspective, ESEM '20, October 8–9, 2020, Bari, Italy.
14. Azevedo B., Jino M., Modeling Traceability in Software Development: A Metamodel and a Reference Model for Traceability, ENASE, School of Electrical and Computer Engineering. University of Campinas, Brazil, 8 p.
15. Kuhrmann M., Fernández D., Towards Artifact Models as Process Interfaces in Distributed Software Projects, IEEE workshop proceedings, 10 p.
16. Seichter D., Dhungana D., Pleuss A., Hauptmann B. Knowledge Management in Software Ecosystems: Software Artefacts as First-class Citizens, ECSA 2010 August 23–26, 2010. Copenhagen. Denmark. P. 119–126.
17. Sadi M., Yu E. Designing Software Ecosystems: How Can Modeling Techniques Help? Springer-Verlag, Berlin Heidelberg. 2015. 15 p.
18. Сидоров Н.А. Экология программного обеспечения. *Інженерія програмного забезпечення*. 2010. № 1. С. 53–61.
19. .Yu E. Modelling Strategic Relationships for Business Process Reengineering. Ph.D., thesis. Dept. of Computer Science, University of Toronto. 1995.
20. Knodel J., Manikas K. Towards a typification of software ecosystems. In Fernandes et al. Software Business – 6th International Conference. ICSOB 2015. Braga, Portugal. June 10–12, 2015. Proceedings 2015. vol. 210 of Lecture Notes in Business Information Processing. Springer. P. 60–65.
21. Сидоров Н.А. Стилистика программного обеспечения. *Проблеми програмування*. 2018. 2, 3. С. 245–254.
22. Sidorov N., Sidorova N., Pirog A. Ontology-driven tool for utilizing programming styles. *Вісник НАУ*. 2017. Том 71. № 2. С. 84–93.
23. Sydorov N., Sydorova N., Sydorov E., Cholyshkina O., Batsurovska I. Development of the approach to using a style in software engineering. *Eastern-European Journal of Enterprise Technologies*. 2019. 4/2 (100). P. 41–51.
24. Сидоров Н.А., Сидорова Н.Н., Сидоров Е.Н. (2020) Дескриптивная модель экосистемы стиля программирования, Проблеми програмування, Спеціальний випуск, Матеріали конференції, УкрПрог. 2020. № 2-3. С. 74–81.

Received 04.11.2020

About author:

Sydorov Nikolay,
Doctor of Technical Sciences, Professor,
Number of scientific publications in Ukrainian publishing houses – 105.
Number of scientific publications in foreign publishing houses – 35.
https://orcid.org/0000-0002-3794-780X.

Affiliation:

National Technical University of Ukraine Igor Sikorsky Kyiv Polytechnic Institute,
Department of Automated Information Processing and Control Systems,
03056, Kiev - 56, Prosp. Peremohy, 37.

E-mail: nyksydorov@gmail.com