UDC 004.051 **DOI: 10.20998/2079-0023.2021.02.11**

A. M. KOPP, D. L. ORLOVSKYI, D. ERSOYLEYEN

AN APPROACH TO ANALYSIS OF ARCHIMATE APPLICATION ARCHITECTURE MODELS USING THE SOFTWARE COUPLING METRIC

Applications architecture is the baseline of any organizational activity, which main goal is to provide the executional environment for business processes in order to deliver products or services to satisfy customer needs and generate revenue. Nowadays, large software engineering projects always begin with the architecture design phase, despite the waterfall or agile methodology is used by a software development team. Applications architecture design is the most important and, at the same time, error-prone stage of the whole software engineering project. It is well-known that design shortcomings made on the design phase may increase drastically to testing and maintenance phases. Further costs to defects fixing may be hundred times higher in the later project stages in compare to the design stage on which applications architecture is defined. Common system design solutions, which were proven on practice and used in many projects, are known as architectural patterns. Software architecture patterns are considered as building block for system implementation. The most popular and efficient way to share architectural patterns are graphical models that used as any other blueprints of engineering solutions. Applications architecture models are built to represent system design, whereas, such models are already based on certain patterns as the industry best practices. Hence, in this paper we consider a relevant problem of applications architecture models analysis, which relevance is defined by those fact that designed blueprints of information systems and other software solutions should be carefully checked for all presumable inefficiencies in order to avoid extra efforts and related costs for defects fixing in the later project stages. It is proposed to use ArchiMate enterprise architecture modeling language, since it can be used not only to represent applications architecture, but is connection to business and technology layers. In order to evaluate applications architecture models, respective ArchiMate metamodel is considered and represented as labeled directed graph, and coupling software metric is selected for analysis. Sample calculations are demonstrated, obtained results are discussed, conclusion and future work directions are formulated.

Keywords: applications architecture, software engineering, software design, model analysis, software metrics, coupling metric.

А. М. КОПП, Д. Л. ОРЛОВСЬКИЙ, Д. ЕРСОЙЛЕЄН

ПІДХІД ДО АНАЛІЗУ МОДЕЛЕЙ АРХІТЕКТУРИ ЗАСТОСУНКІВ ARCHIMATE З ВИКОРИСТАННЯМ МЕТРИКИ ЗВ'ЯЗНОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Архітектура застосунків є основою будь-якої організаційної діяльності, головною метою якої є забезпечення середовища для виконання бізнес-процесів з метою надання продуктів або послуг для задоволення потреб клієнтів і отримання прибутку. На сьогоднішній день великі проекти програмної інженерії завжди починаються з фази проектування архітектури, незважаючи на те, що команда розробників програмного забезпечення використовує водоспад або ж гнучку методологію. Проектування архітектури застосунків є найважливішим і, водночас, найбільш вразливим до помилок етапом усього проекту з розробки програмного забезпечення. Добре відомо, що значущість недоліків, припущених на етапі проектування, може різко збільшитися при переході до етапів тестування та супроводу. Подальші витрати на виправлення дефектів можуть бути у сто разів вищими на більш пізніх стадіях проекту, ніж на етапі проектування, на якому визначається архітектура застосунків. Поширені рішення з проектування систем, які перевірені на практиці і використовуються в багатьох проектах, відомі як архітектурні шаблони. Шаблони архітектури програмного забезпечення розглядаються як будівельні блоки для реалізації усієї системи. Найпопулярнішим і ефективним способом обміну архітектурними шаблонами є графічні моделі, які використовуються як і будь-які інші креслення інженерних рішень. Моделі архітектури застосунків створені для представлення проектів системи, причому такі моделі вже базуються на певних шаблонах як найкращих галузевих практиках. Отже, в даній роботі розглядається актуальна проблема аналізу моделей архітектури застосунків, важливість якої визначається тим, що розроблені проекти інформаційних систем та інших програмних рішень повинні бути ретельно перевірені на наявність усіх імовірних неефективних рішень для того, щоб уникнути додаткових зусиль і пов'язаних з цим витрат, спрямованих на усунення дефектів на пізніх стадіях проекту. Пропонується використовувати мову моделювання архітектури підприємства ArchiMate, оскільки її можна використовувати для представлення не тільки архітектури застосунків, а і її зв'язку з рівнями бізнесу та технологій. Для аналізу моделей архітектури застосунків була розглянута відповідна метамодель ArchiMate та здійснено її подання у вигляді розміченого орієнтованого графа, а також для аналізу було обрано метрику зв'язності програмного забезпечення. Продемонстровано приклад розрахунків та проаналізовано отримані результати, сформовано висновки та визначено напрямки подальшої роботи.

Ключові слова: архітектура застосунків, програмна інженерія, проектування програмного забезпечення, аналіз моделей, метрики програмного забезпечення, метрика зв'язності.

А. М. КОПП, Д. Л. ОРЛОВСКИЙ, Д. ЭРСОЙЛЕЕН

ПОДХОД К АНАЛИЗУ МОДЕЛЕЙ АРХИТЕКТУРЫ ПРИЛОЖЕНИЙ ARCHIMATE С ИСПОЛЬЗОВАНИЕМ МЕТРИКИ СВЯЗАННОСТИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Архитектура приложений является основой любой организационной деятельности, главной целью которой является обеспечение среды для выполнения бизнес-процессов с целью предоставления продуктов или услуг для удовлетворения потребностей клиентов и получения прибыли. На сегодняшний день крупные проекты программной инженерии всегда начинаются с фазы проектирования архитектуры, несмотря на то, что команда разработчиков программного обеспечения использует водопад или же гибкую методологию. Проектирование архитектуры приложений является важнейшим и одновременно наиболее уязвимым к ошибкам этапом всего проекта по разработке программного обеспечения. Хорошо известно, что значимость недостатков, предполагаемых на этапе проектирования, может резко увеличиться при переходе к этапам тестирования и сопровождения. Дальнейшие затраты на устранение дефектов могут быть в сто раз выше на более поздних стадиях проекта, чем на этапе проектирования, на котором определяется архитектура приложений. Распространенные решения по проектированию систем, проверенных на практике и используемых во многих проектах, известны как архитектурные шаблоны. Шаблоны архитектуры программного обеспечения рассматриваются как строительные блоки для реализации всей системы. Самым популярным и эффективным способом обмена архитектурными шаблонами являются графические модели, которые используются как и любые другие чертежи инженерных решений. Модели архитектуры приложений предназначены для представления проектов системы, причем такие модели уже базируются на определенных шаблонах как лучших отраслевых практиках. Следовательно, в данной работе рассматривается актуальная проблема анализа моделей архитектуры приложений, важность которой определяется тем, что разработанные проекты информационных систем и других программных решений должны быть тщательно проверены на наличие всех вероятных неэффективных

решений для того, чтобы избежать дополнительных усилий и связанных с этим расходов, направленных на устранение дефектов на поздних стадиях проекта. Предлагается использовать язык моделирования архитектуры предприятия ArchiMate, поскольку его можно использовать для представления не только архитектуры приложений, но и ее связи с уровнями бизнеса и технологий. Для анализа моделей архитектуры приложений была рассмотрена соответствующая метамодель ArchiMate и осуществлено ее представление в виде размеченного ориентированного графа, а также для анализа была выбрана метрика связанности программного обеспечения. Продемонстрирован пример расчетов и проанализированы полученные результаты, сформированы выводы и определены направления дальнейшей работы.

Ключевые слова: архитектура приложений, программная инженерия, проектирование программного обеспечения, анализ моделей, метрики программного обеспечения, метрика связанности.

Introduction. Application architecture designs can be evaluated to ensure that quality attributes are met. Pre-implementation architectural approaches are used by system architects during the initial design and preparation stages before actual implementation begins. In contrast to implementation-oriented architecture compliance approaches, it is assessed whether the implemented system architecture matches the intended system architecture. Architectural conformance approach evaluates whether the implemented architecture is consistent with the proposed architecture specification and the objectives of the proposed architecture [1].

Architectural styles, approaches, or techniques are used within the software system design process to evaluate the software architecture in the pre-implementation phase. Approaches or techniques are design decisions that affect the control of the quality attribute response. Architectural styles or patterns describe the structure and interactions between system components [1].

There are software architecture methods in the systems design based on their quality attributes, such as Attribute Driven Design (ADD) [2] (see fig. 1).

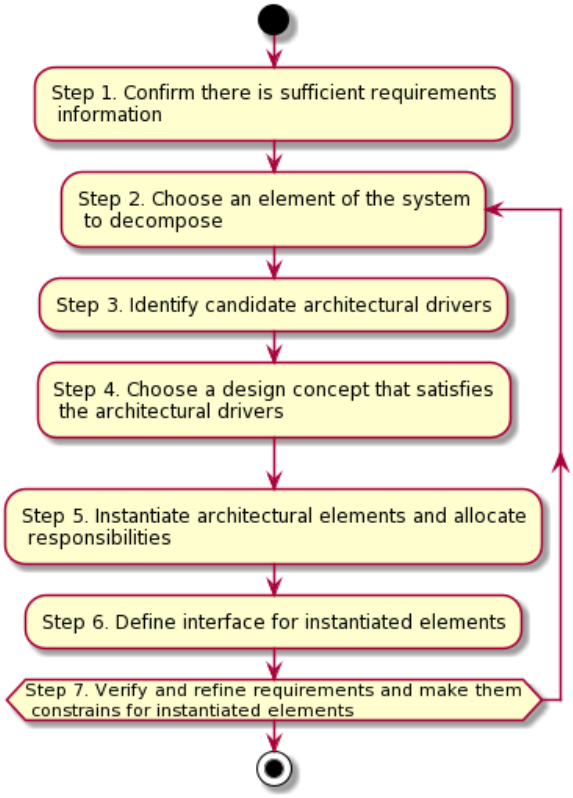

Fig. 1. ADD method steps [9].

The ADD method is an approach to the definition of software architecture, in which the design process is based on software quality attributes. ADD complies with the recursive design process that decomposes the system or system element by applying the architectural tactics and models that satisfy its driving requirements. As shown in fig. 1 above, adding essentially follows the “plan, do, and check” cycle [3]:

- plan: quality attributes and design constraints are considered to select which types of elements will be used in the application architecture;
- do: elements are instantiated to satisfy quality attribute requirements (also referred as nonfunctional requirements) as well as functional requirements;
- check: the resulting design is analyzed to determine if the requirements are met.

This process is repeated until all architecturally significant requirements are met.

Also there are methods for assessing the conformity of quality attributes to software architecture design, such as the Architecture Tradeoff Analysis Method (ATAM) [4] (see fig. 2).

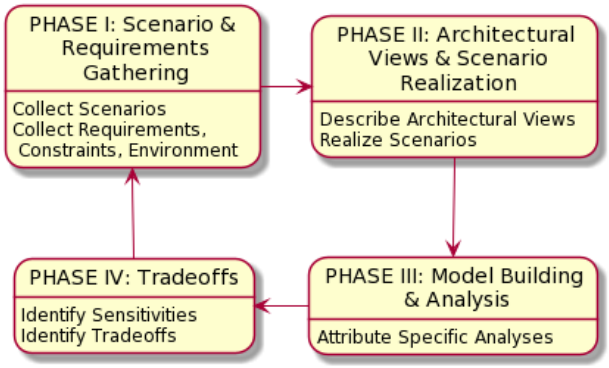

Fig. 2. ATAM method steps [11].

ATAM is a method for evaluating architecture-level designs that considers multiple quality attributes such as modifiability, performance, reliability and security in gaining insight as to whether the fully fleshed out incarnation of the architecture will meet its requirements. The method identifies trade-off points between these attributes, facilitates communication between stakeholders (such as user, developer, customer, maintainer) from the perspective of each attribute, clarifies and refines requirements, and provides a framework for an ongoing, concurrent process of system design and analysis [5].

Problem statement. It is well known that different architectural solutions have their strengths and weaknesses, which may affect development, testing, and maintenance stages of a software system. Therefore, the problem of applications architecture models analysis become relevant, since designed blueprints of the software system should be carefully checked for all presumable inefficiencies in order to avoid extra efforts and related costs for defects fixing in the later project stages.

This study aims on detection of strong and weak spots of software design solutions by analyzing applications architecture models. Research objective includes the process of applications architecture models analysis. Research subject considers the method for applications architecture models analysis. In order to achieve research goal, there should be selected applications architecture modeling language, defined its metamodel, and proposed measures for structural analysis of designed models.

Materials and methods. Methods ADD and ATAM follow a recursive process based on the quality attributes that the system must meet. At each stage, techniques and architectural patterns (or styles) are selected in case they satisfy certain qualities [1].

More holistic approach to architecture design and analysis (fig. 3) is proposed by the TOGAF (The Open Group Architecture Framework) [6]. The Architecture Development Methodology (ADM) is used to develop enterprise architecture (EA), which meets the needs of the organization of business and information technologies (IT).

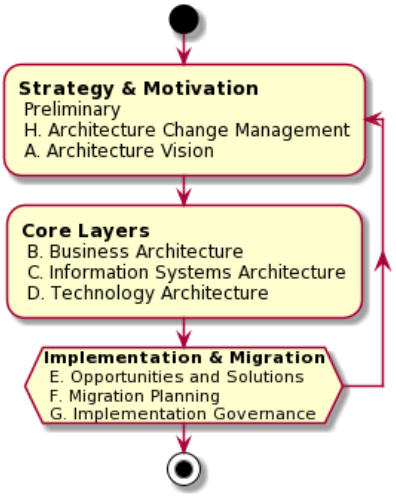

Fig. 3. TOGAF ADM method steps [13].

TOGAF is a mature approach for enterprise architecture and framework used by leading world organizations to improve business efficiency. This is the most outstanding and reliable architecture of the enterprise to ensure consistency of standards, methods and communications between professionals of the enterprise of architecture. Professionals working in TOGAF standards enjoy a higher sectorial reputation, efficiency and career capabilities. TOGAF helps practitioners, avoid entering patented methods, achieve more efficient use of resources and achieve a higher return on investment [6].

IT (Information Technology) architecture should carefully reflect the business goals of the organization. In fact, specific technologies (business scenarios) should be used to ensure that it is to correctly understand the business goals and are reflected in the IT architecture designed using TOGAF [6].

ADM is the result of a constant contribution of a large number of practicing architecture in the following goals [6]:

- it describes a method for developing and managing the life cycle of EA and forms the TOGAF core;
- it can be configured in accordance with the needs of the organization, and then used to implement measures for planning the management system structure.

Even though TOGAF ADM is not that different from the previously considered ADD and ATAM approaches, especially considering the cyclic nature of all of referred methods including ADM, it has significant advantages:

- ADM considers business architecture, strategy, and implementation activities, focusing not only on the software quality and functional attributes, but also taking into account the real business needs of customers therefore providing more holistic and reliable approach to application and IT architecture development;
- ADM could be formalized using the architectural modeling language proposed by TOGAF – ArchiMate [7].

In order to ensure a single presentation of architectural descriptions, the ArchiMate modeling language was developed, offering an integrated approach to the description and visualization of various organizational regions, their relationships and dependencies. The aim of the ArchiMate project is to provide architects to support tools and improve the process of developing an enterprise architecture. Currently, ArchiMate is the Open Group standard. Organizational areas in ArchiMate are associated with the help of a service-oriented paradigm, where each layer provides the functionality of the preceding layer in the form of services. As a formal language of visual design, ArchiMate supports different points of view for individual stakeholders, and is quite flexible for subsequent expansion. For example, for a more complete covering of the TOGAF methodology, in the second version of the ArchiMate language, were introduced new viewpoints – Motivation Extension and Implementation and Migration Extension (see fig. 3) [7].

Metamodel of ArchiMate active structure applications architecture elements, as well as their inter-relationships are demonstrated in fig. 4. All of the proposed elements and relationships are sufficient to design application and IT architectures.

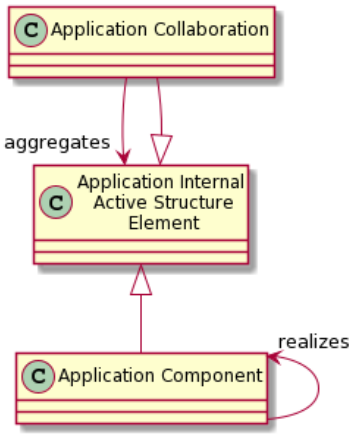

Fig. 4. Metamodel of ArchiMate active structure applications architecture elements [8].

In general ArchiMate metamodel [8] includes two main types of elements:

- structure elements that can be subdivided into active structure elements and passive structure elements: active structure elements can be further subdivided into external active structure elements (also called interfaces) and internal active structure elements;
- behavior elements that can be subdivided into internal behavior elements, external behavior elements (also called services), and events.

Let us clarify on this ArchiMate behavioral and structural building blocks [8], [9]:

- active structural elements to which interfaces as external active structure elements (e.g. application and generic or domain-specific internal active structure elements (e.g. application components) belong;
- behavior elements include services as external behavior elements (e.g. application services) and generic or domain-specific internal behavior elements (e.g. application functions);
- passive structure elements are structural elements that cannot perform behavior, they are often information or data objects, but they can also represent physical objects.

Results. Since ArchiMate language has its own specification and metamodel, architectures of application and IT systems described using this language could be formally described.

Generic framework for EA modeling [9] focused on behavior, active structure, and passive structure elements demonstration is shown in fig. 5.

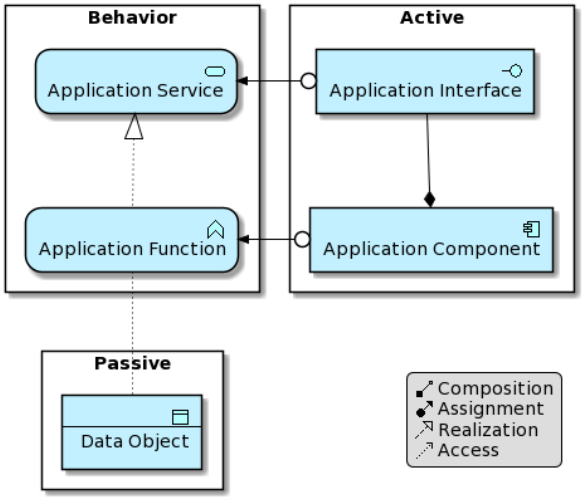

Fig. 5. Essential ArchiMate cross-layer reference model [9].

Outlined reference model demonstrates not only elements of different prospects (passive structure and behavior) together with active structure elements mentioned in fig. 4 earlier, but also possible types of relationships between such elements (e.g. composition, assignment, realization, and access) [8], [9].

In general ArchiMate model could be formally described using the following tuple [10]:

$$AM = \langle V, E, C, R, vc, er \rangle, \qquad (1)$$

here V – is the set of vertices (define and describe architectural elements of applications and IT);

$E \subset V \times V$ – is the set of edges (define and describe relationships between architectural elements);

C – is the set of types of architectural elements defined by the ArchiMate metamodel;

R – is the set of types of relationships between architectural elements;

$vc: V \to C$ – is the function that maps types of architectural elements to the vertices of the graph AM (1);

$er: E \to R$ – is the function that maps types of relationships to the edges of the graph AM (1).

Hence, using this reference model (see fig. 5) and equation (1) for the first time shown in [10], we can easily apply software metrics to evaluate applications architecture models given in ArchiMate language.

A software coupling measure reflects the strength of interconnection between modules by considering incoming and outgoing connections [11]. As we know from software engineering basics [12]:

- "weak" or "low" coupling is a feature of software components that have small amount of external connections (both incoming and outgoing), since they autonomously solve distinct tasks, being efficient for modification, re-using, and testing; "low" coupling is a property of well-structured and properly designed system;
- "strong" or "high" coupling in contrast could be considered as a serious shortcoming; "high" coupling is a of bad-structured and poorly designed systems, which are hard for understanding and modification, while distinct components cannot be autonomously tested and re-used.

Therefore, we can formulate the following equation in order to calculate coupling of each applications architecture components:

$$C(v) = 1 - \frac{1}{1 + \mathrm{d}_{in}(v) + \mathrm{d}_{out}(v)}, v \in V, \qquad (2)$$

here $\mathrm{d}_{in}(v)$ – is the number of incoming connections of a certain architecture component $v \in V$;

$\mathrm{d}_{out}(v)$ – is the number of outgoing connections of a certain architecture component $v \in V$.

Proposed metric (2) ranges approximately from 0 for "weak" coupling, see fig. 6, to 1 for "strong" coupling, see equations (3) and (4).

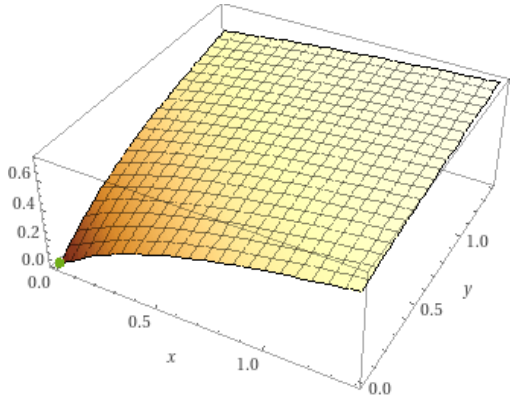

Fig. 6. Minimum of $C(v)$ with $\mathrm{d}_{in}(v) \geq 0$ and $\mathrm{d}_{out}(v) \geq 0$

$$\lim_{\mathrm{d}_{in}(v)\to+\infty}\left(1-\frac{1}{1+\mathrm{d}_{in}(v)+\mathrm{d}_{out}(v)}\right)=1, \quad (3)$$

$$\lim_{\mathrm{d}_{out}(v)\to+\infty}\left(1-\frac{1}{1+\mathrm{d}_{in}(v)+\mathrm{d}_{out}(v)}\right)=1. \quad (4)$$

Let us also provide the following generalized metric that could be used in order to measure coupling of the whole applications architecture rather than for a distinct software component at is done by (2):

$$C_{avg}=\frac{1}{|V|}\cdot\sum_{v\in V}C(v)\,,C_{max}=\max_{v\in V}C(v), \quad (5)$$

here C_{avg} – is the generalized coupling measure that can be used if compensation of certain poor-structured application components of "high" coupling by other well-structured application components of "low" coupling is allowed;

C_{max} – is the generalized coupling measure that can be used if compensation of bad applications architecture decisions by good ones as it is mentioned above for C_{avg} is denied to achieve modifiable, maintainable, and re-usable software systems.

As well as (2), proposed generalized metric (5) ranges approximately from 0 for "weak" coupling to 1 for "strong" coupling of the whole applications architecture.

In order to verify proposed coupling metrics, let us analyze web presentation patterns presented in Martin Fowler's "Catalog of Patterns of Enterprise Application Architecture" [13]:

- Model View Controller (MVC);
- Page Controller (PC);
- Front Controller (FC);
- Template View (TPV);
- Transform View (TFV);
- Two-Step View (TSV);
- Application Controller (AC).

Results obtained for each of these web presentation enterprise application architecture patterns are shown in table 1 below.

Table 1 – Coupling of web presentation patterns.

Pattern	Size	Min. $C(v)$	Max. $C(v)$	Avg. $C(v)$
MVC	3	0.67	0.80	0.71
PC	3	0.00	0.80	0.49
FC	4	0.00	0.86	0.21
TPV	3	0.00	0.67	0.44
TFV	3	0.00	0.67	0.44
TSV	5	0.00	0.80	0.43
AC	4	0.67	0.80	0.70

Analysis and discussion. According to the obtained results (see table 1 above), the "Model View Controller" web presentation pattern demonstrates the "highest" coupling, while the "Front Controller" web presentation pattern (which is considered as implementation of MVC pattern) shows the "lowest" coupling among application architecture components.

Therefore, the "Front Controller" pattern could be recommended as the best solution for enterprise web application architecture design because of its low coupling and, hence, better modifiability, maintainability, and re-usability.

Obtained results are proved by almost twenty years of MVC and, in particular, FC dominance in enterprise applications development (see table 2) thank to its concept of never mixing data with presentation.

Table 2 – MVC-based development frameworks support [14].

Language	Frameworks
PHP	CodeIgniter, Laravel, Symfony, Yii, Zend etc.
Java	Spring
Python	Django, Flask
JavaScript	Angular, Express, React etc.
.NET	ASP.NET, Silverlight

Conclusion and future work. In order to summarize, we need to state that ADM and ArchiMate language are more preferable ways to describe and analyze software application and IT system architectures rather than ADD and ATAM methods, since the TOGAF baseline of ArchiMate considers all of the valuable aspects of customer's business giving more holistic description that takes into account not only software attributes, but also their connections to the goals and strategy of a particular organization, which requires improvement through IT services implementation.

Introduced formalisms (1), (2), and (5) could be used to process ArchiMate models metadata in order to analyze applications architecture domain models, identify poorly-designed architecture fragments, and resolve inefficiencies in order to avoid further software implementation, testing, and maintenance errors, as well as related expenses caused by "strongly" coupled application components that cannot be properly modified, tested, and re-used [12].

Moreover, graph-based description of applications architecture models allows to use propagation cost analysis (i.e. which percent of all IT landscape will be affected by error fixing or other re-design efforts) used earlier in [15] for business architecture.

Future work in this field includes extension of the proposed approach in order to consider other ArchiMate enterprise architecture domains, such as business and technology layers, as well as information technology design and development in order to implement proposed approach as a tool for practicing system and software architects, researchers, and other stakeholders.

References

1. Tekinerdogan B. et al. Quality concerns in large-scale and complex software-intensive systems. Software Quality Assurance. *Large Scale and Complex Software-Intensive Systems*. 2016. P. 1–17.
2. *Attribute Driven Design*. URL: http://safordevs.blogspot.com/2015/07/attribute-driven-design.html (accessed 10.02.2021).
3. Wojcik R. et al. *Attribute-Driven Design (ADD), Version 2.0*. Carnegie Mellon University, 2006. 55 p.
4. *Making Tradeoffs and Choices*. URL: https://www.evolute.be/thoughts/atam.html (accessed 18.02.2020).
5. Kazman R. et al. The Architecture Tradeoff Analysis Method. *Proceedings of ICECCS98*. 1998. P. 1–11.
6. *Read TOGAF Enterprise Architecture*. URL: https://developpaper.com/read-togaf-enterprise-architecture (accessed 03.03.2021).

7. Vicente M. et al. Using ArchiMate and TOGAF to Understand the Enterprise Architecture and ITIL Relationship. *Lecture Notes in Business Information Processing*. 2013. No. 148. P. 134–145.
8. *ArchiMate Specification*. URL: https://pubs.opengroup.org/architecture/archimate3-doc/chap04.html#_Toc10045299 (accessed 04.03.2021).
9. Aulkemeier F. et al. A Service-Oriented E-Commerce Reference Architecture. *Journal of Theoretical and Applied Electronic Commerce Research*. 2016. Vol. 11. No. 1. P. 26–45.
10. Klimek R., Szwed P. Verification of ArchiMate process specifications based on deductive temporal reasoning. *Federated Conference on Computer Science and Information Systems*. 2013. P. 1103–1110.
11. Ingeno J. *Software Architect's Handbook: Become a successful software architect by implementing effective architecture concepts*. Packt Publishing Ltd, 2018. 594 p.
12. *Software Engineering – Coupling and Cohesion*. URL: https://www.geeksforgeeks.org/software-engineering-coupling-and-cohesion/ (accessed 20.04.2021).
13. *Catalog of Patterns of Enterprise Application Architecture*. URL: https://martinfowler.com/eaaCatalog/ (accessed 26.04.2021).
14. *Languages and Frameworks for Programming in 2021*. URL: https://www.whoishostingthis.com/compare/languages-and-frameworks/ (accessed 28.04.2021).
15. Orlovskyi D., Kopp A. Enterprise architecture modeling support based on data extraction from business process models. *CEUR Workshop Proceedings*. 2020. Vol. 2608. P. 499–513.

References (transliterated)

1. Tekinerdogan B. et al. Quality concerns in large-scale and complex software-intensive systems. Software Quality Assurance. *Large Scale and Complex Software-Intensive Systems*. 2016, pp. 1–17.
2. *Attribute Driven Design*. Available at: http://safordevs.blogspot.com/2015/07/attribute-driven-design.html (accessed 10.02.2021).
3. Wojcik R. et al. *Attribute-Driven Design (ADD), Version 2.0*. Carnegie Mellon University, 2006. 55 p.
4. *Making Tradeoffs and Choices*. Available at: https://www.evolute.be/thoughts/atam.html (accessed 18.02.2020).
5. Kazman R. et al. The Architecture Tradeoff Analysis Method. *Proceedings of ICECCS98*. 1998, pp. 1–11.
6. *Read TOGAF Enterprise Architecture*. Available at: https://developpaper.com/read-togaf-enterprise-architecture (accessed 03.03.2021).
7. Vicente M. et al. Using ArchiMate and TOGAF to Understand the Enterprise Architecture and ITIL Relationship. *Lecture Notes in Business Information Processing*. 2013, no. 148, pp. 134–145.
8. *ArchiMate Specification*. Available at: https://pubs.opengroup.org/architecture/archimate3-doc/chap04.html#_Toc10045299 (accessed 04.03.2021).
9. Aulkemeier F. et al. A Service-Oriented E-Commerce Reference Architecture. *Journal of Theoretical and Applied Electronic Commerce Research*. 2016, no. 1 (11), pp. 26–45.
10. Klimek R., Szwed P. Verification of ArchiMate process specifications based on deductive temporal reasoning. *Federated Conference on Computer Science and Information Systems*. 2013, pp. 1103–1110.
11. Ingeno J. *Software Architect's Handbook: Become a successful software architect by implementing effective architecture concepts*. Packt Publishing Ltd, 2018. 594 p.
12. *Software Engineering – Coupling and Cohesion*. Available at: https://www.geeksforgeeks.org/software-engineering-coupling-and-cohesion/ (accessed 20.04.2021).
13. *Catalog of Patterns of Enterprise Application Architecture*. Available at: https://martinfowler.com/eaaCatalog/ (accessed 26.04.2021).
14. *Languages and Frameworks for Programming in 2021*. Available at: https://www.whoishostingthis.com/compare/languages-and-frameworks/ (accessed 28.04.2021).
15. Orlovskyi D., Kopp A. Enterprise architecture modeling support based on data extraction from business process models. *CEUR Workshop Proceedings*. 2020, vol. 2608, pp. 499–513.

Надійшла (received) 20.06.2021

Відомості про авторів / Сведения об авторах / About the Authors

Копп Андрій Михайлович – доктор філософії, Національний технічний університет «Харківський політехнічний інститут», доцент кафедри програмної інженерії та інформаційних технологій управління; м. Харків, Україна; ORCID: http://orcid.org/0000-0002-3189-5623; e-mail: kopp93@gmail.com

Орловський Дмитро Леонідович – кандидат технічних наук, доцент, Національний технічний університет «Харківський політехнічний інститут», доцент кафедри програмної інженерії та інформаційних технологій управління; м. Харків, Україна; ORCID: http://orcid.org/0000-0002-8261-2988; e-mail: orlovskyi.dm@gmail.com

Ерсойлеєн Дорукхан – Національний технічний університет «Харківський політехнічний інститут», студент кафедри програмної інженерії та інформаційних технологій управління; м. Харків, Україна; e-mail: ersoyleyen_1@hotmail.com

Копп Андрей Михайлович – доктор философии, Национальный технический университет «Харьковский политехнический институт», доцент кафедры программной инженерии и информационных технологий управления; г. Харьков, Украина; ORCID: http://orcid.org/0000-0002-3189-5623; e-mail: kopp93@gmail.com

Орловский Дмитрий Леонидович – кандидат технических наук, доцент, Национальный технический университет «Харьковский политехнический институт», доцент кафедры программной инженерии и информационных технологий управления; г. Харьков, Украина; ORCID: http://orcid.org/0000-0002-8261-2988; e-mail: orlovskyi.dm@gmail.com

Эрсойлеен Дорукхан – Национальный технический университет «Харьковский политехнический институт», студент кафедры программной инженерии и информационных технологий управления; г. Харьков, Украина; e-mail: ersoyleyen_1@hotmail.com

Kopp Andrii Mykhailovych – PhD in Computer Sciences, National technical university «Kharkiv polytechnic institute», Associate Professor of the Department of Software Engineering and Management Information Technologies; Kharkiv, Ukraine; ORCID: http://orcid.org/0000-0002-3189-5623; e-mail: kopp93@gmail.com

Orlovskyi Dmytro Leonidovych – PhD in Technical Sciences, Docent, National technical university «Kharkiv polytechnic institute», Associate Professor of the Department of Software Engineering and Management Information Technologies; Kharkiv, Ukraine; ORCID: http://orcid.org/0000-0002-8261-2988; e-mail: orlovskyi.dm@gmail.com

Ersoyleyen Dorukhan – National technical university «Kharkiv polytechnic institute», Student of the Department of Software Engineering and Management Information Technologies; Kharkiv, Ukraine; e-mail: ersoyleyen_1@hotmail.com