DOI: 10.15514/ISPRAS-2022-34(1)-2

Requirements traceability as the basis for designing a functional and logical architecture of a software system

[1,2] *B.A. Pozin, ORCID: 0000-0002-0012-2230 < bpozin@ec-leasing.ru >*
[3] *G.N. Tsiperman, ORCID: 0000-0001-7740-5629 <gntsip@gmail.com>*

[1] *National Research University Higher School of Economics,*
20, Myasnitskaya Ulitsa, Moscow, 101978, Russia
[2] *EC-leasing,*
125 Varshavskoye shosse, Moscow, 117405, Russia
[3] *National University of Science and Technology "MISIS"*
Leninskiy Prospekt 4, Moscow, 119049, Russia

Abstract. The paper deals with the formation and transformation of stakeholder requirements for the information system throughout the entire life cycle. It is shown how the seamless architecture provides traceability of requirements from the level of the business process, to the functional and logical architectures of systems, to the selection of criteria and identification of microservices. It is shown how maintaining the traceability of requirements in the presence of business, functional and logical architecture models can reduce the cost of planning complex functional and load testing of systems, as well as ensure the interaction of operation, maintenance services and contractors that form the entire system, maintain its integrity during the life cycle.

Keywords: information system; requirements engineering; requirements traceability; adaptive clustering method; seamless design; microservice criteria; business architecture; functional architecture

For citation: Pozin B.A., Tsiperman G.N. Requirements traceability as the basis for designing a functional and logical architecture of a software system. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 1, 2022, pp. 23-34. DOI: 10.15514/ISPRAS–2022–34(1)–2

Acknowledgements. The article is based on the materials of the report at the Seventh International Conference «Actual Problems of System and Software Engineering» (APSSE 2021).

Прослеживаемость требований как основа проектирования функционально-логической архитектуры программной системы

[1,2] *Б.А. Позин, ORCID: 0000-0002-0012-2230< bpozin@ec-leasing.ru >*
[3] *Г.Н. Циперман, ORCID: 0000-0001-7740-5629 <gntsip@gmail.com>*

[1] *НИУ Высшая школа экономики,*
101978, Россия, г. Москва, ул. Мясницкая, д. 20
[2] *ЕС-лизинг*
117587, Россия, г. Москва, Варшавское шоссе, д. 125, стр.1
[3] *Национальный исследовательский технологический университет «МИСИС»*
119049, Москва, Ленинский пр-кт, д. 4, стр. 1

Аннотация. В статье рассматриваются вопросы формирования и трансформации требований заинтересованных сторон к информационной системе на протяжении всего жизненного цикла.

Показано, как бесшовная архитектура обеспечивает прослеживаемость требований от уровня бизнес-процесса к функциональной и логической архитектурам систем, к выбору критериев и идентификации микросервисов. Показано, как поддержка прослеживаемости требований при наличии моделей бизнес-архитектуры, функциональной и логической архитектур позволяет снизить затраты на планирование комплексного функционального и нагрузочного тестирования систем, а также обеспечить взаимодействие служб эксплуатации, сопровождения и подрядчиков, формирующих систему в целом, сохранять ее целостность в течение всего жизненного цикла.

Ключевые слова: информационная система; разработка требований; прослеживаемость требований; метод адаптивной кластеризации; бесшовный дизайн; критерии микросервиса; бизнес-архитектура; функциональная архитектура

Для цитирования: Позин Б.А., Циперман Г.Н. Прослеживаемость требований как основа проектирования функционально-логической архитектуры программной системы. Труды ИСП РАН, том 34, вып. 1, 2022 г., стр. 23-34. DOI: 10.15514/ISPRAS–2022–34(1)–2

Благодарности. Статья подготовлена по материалам доклада на Седьмой Международной конференции «Актуальные проблемы системной и программной инженерии» (АПСПИ 2021).

1. Introduction

Due to increasing complexity of software systems, an increasing number of stakeholders (customers, users, developers, administrators, maintenance staff) are involved in ensuring their life cycle. This leads to the need to ensure the transfer of knowledge about requirements to the software system and the system itself between stakeholders. The key task of this process is to provide an understanding of how the original business requirements are translated into functional and non-functional requirements, requirements for the deployment and maintenance of the system.

For understanding, it is necessary to provide a model representation of the system architecture at various levels of abstraction, where the initial requirements are determined based on the stakeholders needs at the business level, and then consistently (seamlessly), without loss or distortion, are converted into requirements for the software system.

Thus, three main tasks are formulating that arise at different stages of the system life cycle (creation, commissioning and maintenance), the solution of which will be considered in this work:

- Adequate implementation and completeness of the functionality of the system, which is validated by the end user, easily understood by him.
- Deployment, i.e., adaptability to easy deployment by the operating unit of the owner of the system, while maintaining an understanding of which business automation functions or entities are located on each element of the deployment.
- Maintenance, i.e., ease of understanding of the logical architecture of the system by the end user and the ability to set tasks for changes to counterparties.

The solution of these problems will be considered as the evolution of the requirements for the system: their definition, transformation and modification. In [1], the need to obtain a coherence set of requirements that provides two-way traceability between the requirements for system functions and the sources of these requirements based on the attribution of requirements.

In this paper, the method of adaptive clustering (ACM) of information systems (IS) is used as a tool for managing a set of requirements for the system that is consistent at all levels of abstraction [2]. CM is a top-down design method, where detailed architectural models are built on the basis of decomposition of the higher abstraction level models elements, ensuring their seamless connection.

Seamless connection is a connection in which the process of decomposition does not violate the traceability of transitions between elements of architectural models.

2. Formulation of the problem

To describe the functional requirements for a software system, it is customary to use particularly Use Case diagrams, which provide a good definition of the requirements for the system functions, linking them to specific business needs Use Case diagrams use knowledge about the business process implicitly, reflecting on a variety of formalized use cases, existing separately from them, without giving a complete picture of the possibilities of automating the business process.

To ensure the completeness of the definition of business requirements, it is necessary to obtain such requirements directly from the business process model, at the level of the business architecture. This is how the initial business requirements for the system are determined in the ACM method [2], in which UML activity diagrams and business process decomposition techniques are used to describe them.

At the next level of architecture (functional architecture), the requirements for the functions of the system are determined based on the business requirements accepted by the stakeholders. The functional architecture models the interaction of the system with users, as well as with other external agents. The functional architecture forms the appearance of the system based on the representation of the compositions of the system's dialogues, defines the requirements for the dialogues and specifies the interfaces with external systems.

The modern practice of designing a functional architecture of a system involves a heuristic transition between architectural models of various levels of abstraction: a more detailed functional architecture is built in such a way that technical solutions meet business requirements as much as possible. In doing so, the architect uses his experience and knowledge to build a detailed model, and then proves (or considers it obvious) that the resulting model satisfies the requirements arising from a more abstract architecture.

As a result, there is no obvious transition from the requirements for automated functions defined during the business architecture stage to the requirements for system functions defined by the functional architecture. This means that there is a technological gap («seam») between architectures of different levels of abstraction. The presence of a seam does not allow tracing the relationship between the requirements for the system functions and the requirements for the functions of the business process that need to be automated.

At the logical architecture level, the technological gap does not allow to transparently identify which microservices implement specific business needs [3] declared in the system, do not provide an opportunity at the business architecture or functional architecture level to define the bounded context and microservice aggregates [4], to give “physical meaning" to the components of the microservice architecture, thereby concretizing the system deployment plan that is understandable to the system owner. In this case, the traceability of the initial requirements in the logical architecture of the system is violated.

Finally, the technological gap does not allow to correctly build a comprehensive testing plan for the system, and also complicates the process of localizing errors and making changes to the current implementation of the system.

Summarizing, we can formulate the following problems that this work is devoted to:

1) The emergence of technological gaps (seams) between the levels of decomposition and the need to form and use rules for eliminating seams to ensure a traceable functional decomposition of business requirements for the system.
2) The technological gap between the business architecture and the functional architecture of the system and how to bridge it.
3) Lack of traceability of requirements in the design of a microservice architecture, considered as a set of units of development, deployment and maintenance.
4) The impossibility of using tracked functional models for planning complex testing of the system, its regression testing and assessing its performance using load testing methods.

5) Impossibility to use the traceability mechanism when planning work on making changes to the maintenance process.

3. Decomposition as a method of representing a business process model

In ACM, it is the business architecture that is the starting point for the design of IS. Business architecture in the context of IS design is understood as a set of models of automated business processes of an enterprise [5]. In the context of this work, a target business architecture is considered, that is, a model of business processes taking into account the use of an information system. The task is to determine, based on the decomposition of the business process, a complete set of business process functions that can be automated.

Decomposition, as a method of representing a business process model, creates a classification scheme that groups operations in accordance with a system of intermediate results of a business process. Decomposition is not the only way to represent a business process. In the limiting case, it can be described as a sequence of operations leading to the desired result. However, such a description for real processes, where the number of operations is in the hundreds, is difficult to understand and analyze.

It is operations (hereinafter we will call them "business operations") that are the ultimate goal of decomposition and are objective in nature. Business operations are located on the leaves of the decomposition tree, and the rest of the structure of the decomposition tree is intended to define a complete list of business process operations.

3.1 Definition of business process decomposition

A business process is understood as a set of actions in which, on the basis of one or more types of initial data [resources], a result is created that is valuable to the client [6]. To describe the decomposition of a business process, the concepts of business function and business operation are sufficient, which are based on the concept of action (*A*), understood as the transformation (*T*) of an initial resource (*Ir*) into a target product (*Tp*).

Decomposition defines the following elements of a business architecture:

- Business function - an action to obtain an intermediate target product required to solve the task of a business process. The business function is always decomposing.
- Business operation is an action that represents the final result of the decomposition within the description of a business process. The result of the business operation itself is not a decomposable target product, but an executor – a specific role function of personnel.

The transformation function T can include components that are predefined information objects (*Po*) that are part of the transformation mechanism (for example, directories or values of the necessary available constants and variables). Predefined objects are created and modified in other contexts. Thus, the action can be defined by a four:

$$A := (Ir, Po, T, Tp).$$

Initial resource (*Ir*) is an information object of the system that initiates an action:

- defines that part of the context of the current state of the system, which is necessary to obtain the target product of this action;
- can represent one or more input information objects, which are the target products of previous actions that initiate this action.
- Target product (*Tp*) is an information object of the system that represents the result of an action:
- defines a part of the context of the new state of the system, which is necessary for the further advancement of the business process or its completion;

- can be one or more information objects that represent an intermediate result of a business process.

Functional decomposition of an action is an operator ($\widehat{D}$), defined on a set of actions ($\boldsymbol{\mathcal{A}}$), representing an action as a set of action elements that implement the original action:

$$\widehat{D}A \rightarrow (A_1, A_2, \dots, A_n), A_i \in \mathcal{A}.$$

The action elements obtained as a result of the decomposition are interconnected by relations representing the control flows that initiate the decomposition elements (Fig. 1). By the scenario of a business function, we mean a logical composition that includes:

- business function decomposition elements $(A_1, A_2, \dots, A_n)$;
- initial and final script element from a set of decomposition elements $A_{in}, A_{fin} \subset (A_1, A_2, \dots, A_n)$;
- set of binary relationships between elements $R(A_i, A_j) \in (\mathcal{R}, \emptyset)$, where $\mathcal{R}$ - is the set of admissible connections between actions.

3.2 Coherence requirements of the decomposition model

The coherence of the decomposition model of a business process is understood as the absence of contradictions between the elements of the model, compliance with the reality in terms of describing business operations.

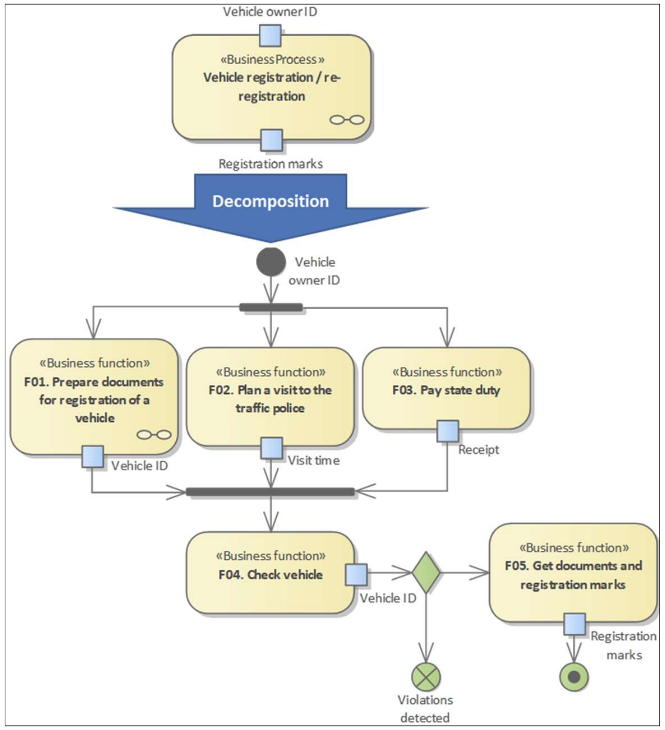

Fig. 1. The first level of a coherent business process decomposition

Two types of coherence are defined:

- *Functional coherence*. Assumes that the action goals of the decomposition elements correspond to the action goals of the decomposed element. To achieve functional consistency, it is necessary to ensure [7]:
  - a) continuity of decomposition - compliance with the sequence of transitions between the levels of the decomposition tree;
  - b) absence of defects - each lower level must reveal the previous one completely, without missing a single element;
  - c) lack of redundancy - it is unacceptable to add during division something that is not in the original.
- *Information (product) coherence*. As an action, the business function converts the *Ir* to *Tp* using the required predefined information objects. The business function decomposition elements, organized into a business function script, should collectively perform the same transformation: the *Ir* at the entrance to the business function decomposition script must match the *Ir* of the decomposed business function, and the final of the script must have a *Tp* that matches the *Tp* of decomposed business function (see fig. 1). Within the script, there must be decomposition elements using the predefined information objects defined for the business function being decomposed. In the scenario itself, new information objects can arise corresponding to the *Ir* and *Tp* of the decomposition elements, and additional predefined information objects can also be used.

Decomposition of a business process allows you to cluster a subject area, dividing it into subdomains corresponding to business functions, with well-defined functional and informational boundaries. Each subdomain represents a bounded context [4], which can be used as the basis for defining a development task for a development team when implementing a microservice architecture of a software system.

4. Formation of microservices

4.1 The task of identifying microservices

In [3], the principle of organizing microservices is declared: the division into services in accordance with the needs of the business. This means that the identification of microservices must be done at the level of the business architecture, i.e., the microservice must have a "physical" meaning. At the level of the architectural (logical) model, there must be an element that defines the boundaries of what we mean by a microservice.

Approaches to the identification of microservices ([8], [9]) mainly dealt with the issues of clustering the IS code according to the criteria of strong and weak connectivity, while simultaneously clarifying the principles that the identified microservices should comply with. The resulting technologies give unstable good results.

The main task for the solution of which the microservice architecture was created is the struggle with the complexity of the development, deployment and maintenance of IS. However, no less important, in our opinion, is the link between the requirements for the system and the system architecture. The key element in this is the concept of a business operation: each business operation must correspond to a microservice that implements its functionality. The input and output of the microservice is determined by the respective Ir and Tp of the business operation. Such a microservice as an architectural element, in our opinion, should correspond to a deployment unit, which will allow tracing in which microservices of the developed and deployed system a particular business operation is implemented, as well as checking the composition of the implemented information context.

This approach requires clustering the functionality and, accordingly, the subject area of the system into maximally independent elements, each of which defines its own limited development context.

This bounded context is determined by the business function. At the same time, there is sufficient freedom in choosing a development unit: the closer a business function is to the root of the decomposition tree, the greater the amount of development it defines.

The independence of development and maintenance objects implies a minimal connection between such objects both in terms of interaction and in terms of using common code: a quick response to changing requirements for one microservice does not imply an order to change the code in another microservice.

With this approach, the criteria for identifying microservices is determined only by business needs, and not by the needs of optimization of development. This is quite consistent with the concept of development and maintenance outlined in [3].

4.2 Criteria for identifying microservices

To determine the criteria for identifying microservices, it is appropriate to give the following analogy. Imagine a car manufacturing industry. Each final product (car) is the result of industry cooperation of various kinds of enterprises that produce both components for assembling a car and the necessary resources. Each enterprise forms its own production infrastructure for the corresponding type of product and, to a reasonable extent, minimizes dependence on related enterprises that are part of the cooperation.

The consistency of the components of the final product is ensured by the necessary standards and contracts between enterprises for the supply of the corresponding component or resource. The contract with the enterprise fixes the terms of delivery of the manufactured product, determined by its name and code within the framework of cooperation.

By this analogy, a microservice is understood as an enterprise included in a cooperation and responsible for the production of one product, and the basis for defining a microservice is a business operation since it meets two main criteria:

- Single Responsibility Principle (SRP) – a business operation is performed under the responsibility of a specific role function of personnel to change the state of one Tp.
- Common Closure Principle (CCP) – The non-decomposable result of a business operation is represented by a single information aggregate (*Tp*).
- The proposed criteria for identifying microservices [10] as deployment units can be formulated as follows:
- When creating an IS, it is difficult to predict which parts of the system will change more often and which less often as a result of operation. To ensure the independence of changes to microservices during operation, the SRP principle can be applied, according to which a microservice should have one responsibility and be associated with one role of an IS user, since the main reason for changes is users.
- Microservice is a software component responsible for changing the state of one specific type of object during its life cycle. In this case, the type of object is an aggregate [4]: a cluster of related objects, which are considered as a whole for the purpose of changing data. External references are limited to one member of the aggregate, designated as the root. This is in line with the CCP principle.

In [8], 16 criteria for the identification of microservices were proposed, concerning, in addition to CCP and SRP, also the issues of determining the development boundaries. Based on a catalog of such criteria, the required system specification artifacts are identified, which can be processed in a structured, semi-automated manner to propose service decomposition that promotes weak communication between services and a high degree of consistency within them. At the same time, the criteria included in the catalog are determined only because, according to the authors, the methods of forming aggregates and defining bounded contexts in the requirements are not known.

5. Seamless transition between business and functional architectures

5.1 Operational Service

The business architecture defines the functional requirements of stakeholders to IS. Business operations are the places where user interactions with the system occur, and therefore the description of business operations allows you to determine the functions that should be automated (automated functions). The set of these functions, defined for a business operation, constitutes the content of the business operation service (hereinafter referred to as the operational service). Operational services are generated for each business operation that includes automated functions. They are architectural elements that are abstract in nature, do not imply any implementation and are intended only to explicitly define the functions to be automated (fig. 2).

An operational service defines the boundary between the business architecture and the functional architecture of the IS. It formalizes the requirements for the automation of business operations and serves as the basis for the formation of technical specifications for the creation or development of IS.

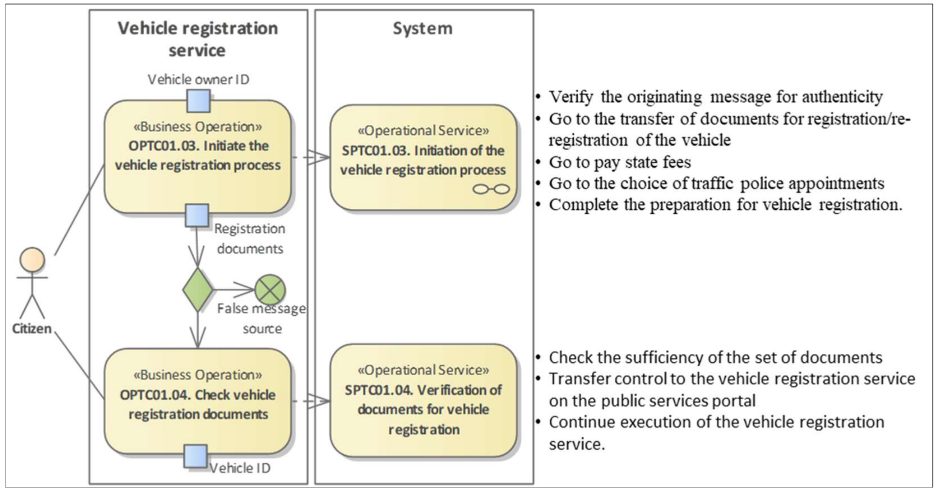
Vehicle registration service
System
Vehicle owner ID
«Business Operation» OPTC01.03. Initiate the vehicle registration process
«Operational Service» SPTC01.03. Initiation of the vehicle registration process
• Verify the originating message for authenticity
• Go to the transfer of documents for registration/re-registration of the vehicle
• Go to pay state fees
• Go to the choice of traffic police appointments
• Complete the preparation for vehicle registration.
Registration documents
False message source
Citizen
«Business Operation» OPTC01.04. Check vehicle registration documents
«Operational Service» SPTC01.04. Verification of documents for vehicle registration
• Check the sufficiency of the set of documents
• Transfer control to the vehicle registration service on the public services portal
• Continue execution of the vehicle registration service.
Vehicle ID

Fig. 2. Business operations, operational services and automated functions (fragment of the diagram)

5.2 Transition to functional architecture

Functional architecture is understood as an architectural representation that includes architectural models of the structure and composition of functional components of the IS that provide access to the "internal" functions of the IS that implement automated functions. In other words, the functional architecture models the interaction of the IS with users, as well as with other external agents. The functional architecture forms the appearance of the IS based on the presentation of the compositions of the system's dialogues, defines the requirements for the dialogues and specifies the interfaces with external systems.

The operational service is the connecting element that defines the seamless connection between the business architecture and the functional architecture: the operational service is decomposed into IS dialogs, the composition of which forms the operational service scenario. Automated functions of a business operation are implemented in dialogs [2].

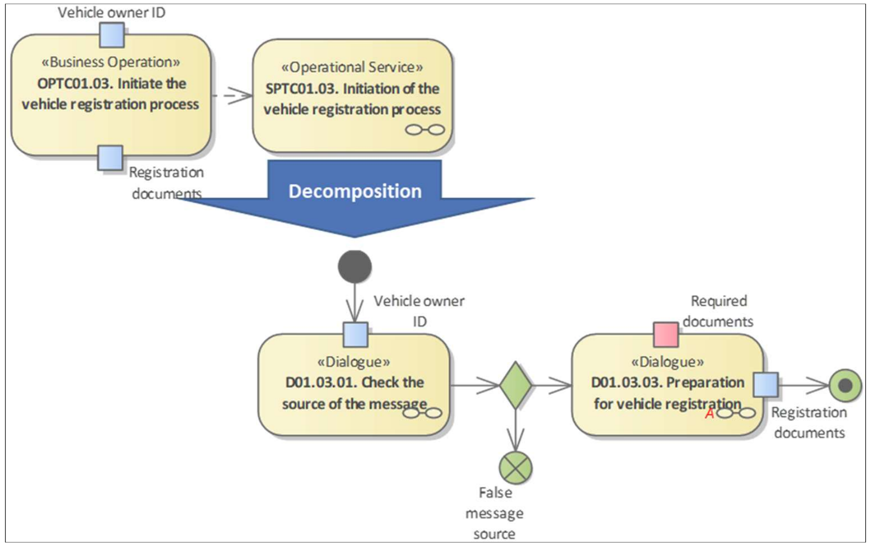
Vehicle owner ID
«Business Operation»
OPTC01.03. Initiate the vehicle registration process
«Operational Service»
SPTC01.03. Initiation of the vehicle registration process
Registration documents
Decomposition
Vehicle owner ID
Required documents
«Dialogue»
D01.03.01. Check the source of the message
«Dialogue»
D01.03.03. Preparation for vehicle registration
Registration documents
False message source

Fig. 3. Seamless transition to functional architecture

In the example (fig. 3), the operational service SPTC01.03 is decomposed into two dialogs D01.03.01 and D01.03.03, which form the service script. The informational coherence of the operational service script with the decomposed service is determined by the fact that the information object "Vehicle owner ID" is passed to the script input, and the "Registration documents" is sent to the output. For dialog D01.03.03, the predefined information object "Required documents" is specified, which is a directory of documents required for registration of vehicles.

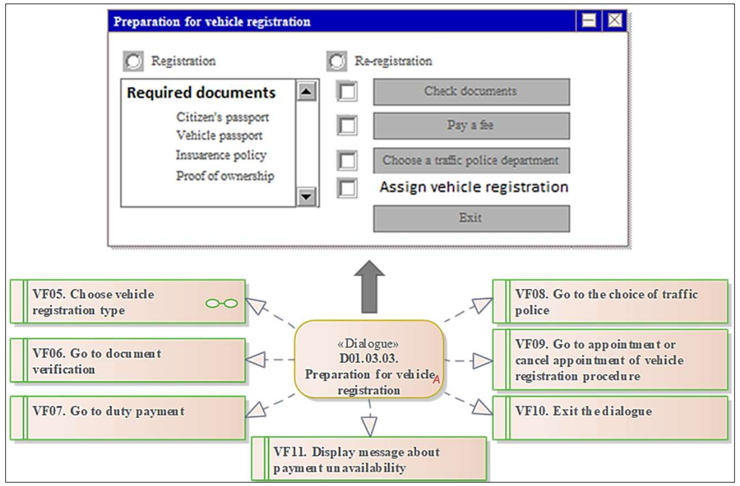
Preparation for vehicle registration
Registration
Re-registration
Required documents
Citizen's passport
Vehicle passport
Insurance policy
Proof of ownership
Check documents
Pay a fee
Choose a traffic police department
Assign vehicle registration
Exit
VF05. Choose vehicle registration type
VF06. Go to document verification
VF07. Go to duty payment
«Dialogue»
D01.03.03.
Preparation for vehicle registration
VF08. Go to the choice of traffic police
VF09. Go to appointment or cancel appointment of vehicle registration procedure
VF10. Exit the dialogue
VF11. Display message about payment unavailability

Fig. 4. Description of the dialog: form and view functions

A system dialogue is understood as any act of interaction between agents that causes a change in the state of the IS by launching the corresponding software components [11]. Thus, the dialogue is understood in a broad sense - it is not only the interaction of the user with the system, but also the exchange of messages between any specified IS objects. Dialogues are described [2]:

- form (if it is a dialogue between the user and the system);
- functions of the IS presentation level (view functions), i.e., functions that provide access to the implementation of automated functions;
- various constraints (preconditions, postconditions, error handling, etc.), which are also considered as view functions (fig. 4).

Since a business operation corresponds to a microservice, then, ultimately, it is the implementation of the operational service that determines its functional and informational boundaries, as well as the interfaces between the IS microservices.

6. Complex system testing

The described approach, based on creating a seamless decomposition from a business process model to a microservice that implements a business operation, while maintaining the traceability of system requirements, makes it possible to build plans for complex testing of systems, including integrated complex testing, on a single basis.

Static complex testing to verify the implementation of system functionality can be planned based on business function models as a basis for designing complex tests, and detailed development of test cases and database tables - based on the analysis of business operation models, taking into account business function scenarios.

A static complex plan [12] is easy to document: based on a business function scenario, it is easy to develop and document test quality criteria. This allows not only to generate tests for a single run, but also, taking into account the traceability of requirements, to accumulate tests for regression complex testing, taking into account the achievable degree of coverage of business function scenarios with complex tests.

Dynamic (load) complex testing also requires planning and building (as rule with the participation of stakeholders) dynamic load testing scenarios [12]. For each scenario of carrying out load testing of the system, the formation of a load model is carried out on the basis of scenarios of business functions, taking into account the assessment of the frequency (or probability) of using each business function in the process of functioning of the system being validated on a model or perspective load. In this case, load flows are determined from the standpoint of business rules - scenarios for the implementation of business functions, and the load model is formed taking into account the frequency of calls of end users and other actors to business operations of the system.

7. Planning for change

The planning of functional changes that are supposed to be introduced into the system, when using the current models of all the described levels, is greatly simplified. When shaping changes in the business process model, even a not very deep analysis allows you to understand which business functions from the existing ones will be affected by the changes or what new business functions will need to be implemented. In both cases, changing the boundaries of the system is explicitly reflected in the business function models and their scenarios.

Likewise, the analysis shows which business operations are subject to change or creation. In this case, the control of the integrity of changes is carried out according to the available documents describing the corresponding models, taking into account the traceability of requirements. Based on the lists of business operations subject to change, an appropriate list of microservices to be changed is determined. Thus, the volume of changes can be quickly assessed down to specific software and information components. The same mechanism allows you to estimate the labor intensity of preparing tests for static complex testing.

8. Conclusion

Integration processes are extremely important for complex custom software systems with a lifetime of 10 years or more. The correct formulation of such processes in the life cycle of the system can significantly reduce the costs of creating, maintaining and developing such systems and significantly affect their integrity and quality. The key issue in solving the issues of ensuring the integrity of systems is to ensure the traceability of requirements at the levels of business processes, the formation of functional and logical architectures of systems up to the selection of criteria and identification of microservices.

The paper shows how to provide a functional decomposition of business requirements, describing it in the UML language, correctly identifying the structured elements of the decomposition. Such a structured approach provides up to the logical architecture model to ensure the preservation of the traceability property of requirements, makes it possible to completely get rid of" seams " not only during the design and development of the system, but also during its operation, maintenance and development. The result of such work makes it possible to support the processes of system operation, its maintenance and development, to provide a fairly simple way to manage the configuration of the system being operated by the personnel of the organization that owns the system.

It is shown how maintaining the traceability of requirements in the presence of business, functional and logical architecture models can reduce the cost of planning complex functional and load testing of systems, as well as ensure the interaction of operation, maintenance services and contractors that form the entire system, maintain its integrity during the life cycle. This interaction using the models described in the work allows you to quickly localize the errors that have appeared and identify microservices that are subject to changes, as well as identify requirements that are violated and require more accurate implementation – starting with business requirements.

References / Список литературы

[1] V.K. Batovrin, B.A. Pozin. Requirements Engineering at a Modern Industrial Enterprise. Software Engineering, vol. 10, no. 3, 2019, pp. 114-124 (in Russian) / В.К. Батоврин, Б.А. Позин. Инженерия требований на современном промышленном предприятии. Программная инженерия, том 10, no. 3, 2019 г., стр. 114–124.

[2] G. Tsiperman. Seamless design of information system architecture based on adaptive clustering method. Proceedings of the 6th International Conference Actual Problems of System and Software Engineering (APSSE 2019), CEUR Workshops, vol. 2514, 2019, pp. 38-44.

[3] James Lewis, Martin Fowler. Microservices. A definition of this new architectural term. 25 March 2014. Available at: https://martinfowler.com/articles/microservices.html.

[4] Eric Evans. Domain-Driven Design Reference. Definitions and Pattern Summaries. Dog Ear Publishing, LLC, 2014, 88 p.

[5] ISO/IEC/IEEE FDIS 29148:2017 Systems and software engineering. Life cycle processes. Requirements engineering. Available at: https://www.iso.org/standard/72089.html.

[6] Michael Hammer, James Champy. Reengineering the Corporation. A manifesto for business revolution, Harper Collins Inc., 1993, 272 p.

[7] I.G. Fedorov. A principles of a process model decomposition. Applied informatics, vol. 11, no. 5(65), pp. 19-30 (in Russian), 2016 / И.Г. Фёдоров. Принципы декомпозиции модели процесса. Прикладная информатика, том 11, no. 5(65), 2016, стр. 19-30.

[8] Michael Gysel, Lukas Kölbener et al. Service Cutter: A Systematic Approach to Service Decomposition. Lecture Notes in Computer Science, vol. 9846, 2016, pp. 185-200.

[9] Shanshan Li, He Zhang et al. A dataflow-driven approach to identifying microservices from monolithic applications. The Journal of Systems and Software, vol. 157, 2019, article no. 110380.

[10] C. Richardson. Microservices Patterns: Microservices Patterns: With examples in Java. Manning, 2019, 520 p.

[11] I. Graham. Object-Oriented Methods. Principles and Practice, Third Edition, Addison-Wesley, 2001. 864 p.

[12] B. Pozin, I. Galakhov. Experience in automated functional and load testing in the life cycle of the mission-critical system Baltic Journal of Modern Computing, vol. 8, no. 2, 2020, pp. 241-258.

Information about authors / Информация об авторах

Boris Aronovich POZIN – Doctor of Technical Sciences, Professor of HSE University, CTO of EC-leasing Co. Expert in Software Engineering field, ESSENCE, Software Systems Performance and Integration Testing, Life Cycle Supporting Systems development, implementation and automation. Author of over 200 publications.

Борис Аронович ПОЗИН – доктор технических наук, профессор НИУ ВШЭ, технический директор ЗАО «ЭК-лизинг». Эксперт в области программной инженерии, ESSENCE, тестирования производительности и интеграции программных систем, разработки, внедрения и автоматизации систем поддержки жизненного цикла. Автор более 200 публикаций.

Grigory Naumovich TSIPERMAN – Information system analysis and design expert. Worked in a number of Russian IT companies. Currently, an independent expert. Being Lead Architect and Technical Supervisor, implemented a number of projects of corporate systems, public administration projects, including the Russian system of execution, production, and issuance of biometric citizen passports. The scope of scientific interest includes the development of information systems and knowledge bases, and project management. The author of over 30 scientific publications.

Григорий Наумович ЦИПЕРМАН – специалист по анализу и проектированию информационных систем. Работал в ряде российских ИТ-компаний. В настоящее время независимый эксперт. В качестве ведущего архитектора и технического руководителя реализовал ряд проектов корпоративных систем, проектов государственного управления, в том числе российской системы оформления, производства и выдачи биометрических паспортов граждан. В сферу научных интересов входит разработка информационных систем и баз знаний, управление проектами. Автор более 30 научных публикаций.