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

Elicitation of functional requirements from the application programming interface documentation for functional testing

[1] *E.A. Gerlits, ORCID: 0000-0002-1747-075X <gerlits@ispras.ru>*
[1] *D.S. Kildishev, ORCID: 0000-0002-1000-0165 <kildishev@ispras.ru>*
[1,2,3,4] *A.V. Khoroshilov, ORCID: 0000-0002-6512-4632 <khoroshilov@ispras.ru>*

[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia,*
[2] *Lomonosov Moscow State University, GSP-1, Leninskie Gory, Moscow, 119991, Russia*
[3] *National Research University, Higher School of Economics 20, Myasnitskaya Ulitsa, Moscow, 101978, Russia*
[4] *Moscow Institute of Physics and Technology (MIPT) 141700, Russia, Moscow region, Dolgoprudny, Campus per., 9*

Abstract. We address a common problem in this paper. The only available documentation for a computer program consists of a user API documentation while we need to identify functional requirements and build test suite to test them. We describe a technique for functional requirements elicitation from the user API documentation. Requirements management tool Requality is exploited in this technique. The tool has been used in several industrial software verification projects.

Keywords: requirements elicitation; requirements extraction; API documentation; Requality; requirements management; functional requirements; requirements markup; requirements catalog.

For citation: Gerlits E.A., Kildishev D.S., Khoroshilov A.V. Elicitation of functional requirements from the application programming interface documentation for functional testing. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 1, 2022, pp. 7-22. DOI: 10.15514/ISPRAS–2022–34(1)–1

Выявление функциональных требований в документации программного интерфейса приложения для функционального тестирования

[1] *Е.А. Герлиц, ORCID: 0000-0002-1747-075X <gerlits@ispras.ru>*
[1] *Д.С. Кильдишев, ORCID: 0000-0002-1000-0165 <kildishev@ispras.ru>*
[1,2,3,4] *А.В. Хорошилов, ORCID: 0000-0002-6512-4632 <khoroshilov@ispras.ru>*

[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25*
[2] *Московский государственный университет имени М.В. Ломоносова, 119991, Россия, Москва, Ленинские горы, д. 1*
[3] *НИУ Высшая школа экономики, 101978, Россия, г. Москва, ул. Мясницкая, д. 20*
[4] *Московский физико-технический институт, 141701, Россия, Московская область, г. Долгопрудный, Институтский пер., 9*

Аннотация. Настоящая работа посвящена решению следующей достаточно распространённой проблемы. Единственной существующей документацией программы является пользовательская документация, описывающая программный интерфейс. Требуется выявить функциональные требования к функциям из программного интерфейса и разработать набор тестов. В работе мы описали метод, руководствуясь которым, можно выявить функциональные требования в пользовательской документации программного интерфейса приложения. Для автоматизации этого метода мы используем инструмент для управления требованиями Requality. Инструмент был использован в нескольких индустриальных проектах по верификации программного обеспечения.

Ключевые слова: выявление требований; извлечение требований; выделение требований; программный интерфейс приложения; Requality; управление требованиями; функциональные требования; разметка требований; каталог требований.

Для цитирования: Герлиц Е.А., Кильдишев Д.С., Хорошилов А.В. Выявление функциональных требований в документации программного интерфейса приложения для функционального тестирования. Труды ИСП РАН, том 34, вып. 1, 2022 г., стр. 7-22. DOI: 10.15514/ISPRAS–2022–34(1)–1

1. Introduction

Many computer programs provide an application programming interface (API). These are operating systems, software libraries and even social networks, messengers and online services.

API is usually specified in a user API documentation. This kind of documentation commonly includes overall description of the computer program, its subsystems and classes, describes class attributes and methods (functions) including attribute types, possible attribute values, method signatures, types of return values and behavior of methods and functions.

The behavior of functions is usually written in a natural language. Unlike the requirements specification the function description in the API documentation is often incomplete and even may include conflicting and ambiguous statements. The reason is that the API documentation is intended for the needs of users while the requirements specification is intended for the needs of software developers who need carefully written complete set of requirements. In addition, the requirements specification document is often reviewed, verified and corrected during the software development life cycle.

In this paper, we suppose the only written source of the requirements is the user API documentation. The task is to create functional tests for a subset of functions from this API. This situation is quite common in practice. In our experience, a high quality functional requirements specification is usually available if it is required by a standard like DO-178C [1] or an error in the software under development can lead to significant losses (consequences).

Software test engineers often design tests by reading and analyzing the API documentation text and do not explicitly build a requirements catalog over the API documentation. This approach can sometimes be justified, for instance, when smoke tests are to be developed. However, a high quality functional testing implies assessment of test completeness. A common test adequacy criterion for functional testing is the percentage of the requirements verified by the tests. Every test should somehow be traced to the requirements it verifies to be able to calculate the test coverage. As the behavior of functions is described in the user API documentation in plain text, an additional layer of requirements is needed in which every requirement is isolated explicitly and has a unique identifier [2]. This layer of requirements over the user API documentation is usually called the requirements catalog.

In addition, standard ISO/IEC/IEEE 29148-2018 [3] defines key requirements characteristics. Some of these characteristics are difficult to check in plain texts. These are completeness, verifiability and traceability.

In this paper, we publish a technique to build a catalog of functional requirements over the API documentation. Functional requirements are not explicitly listed in the API documentation as it is in the functional requirements specification. Therefore, our technique aims at elicitation of functional requirements from the API documentation. As we already explained the API documentation may contain ambiguous and conflicting statements and other problems typical for plain texts. We address these challenges in our technique.

This paper is structured as follows. We explain our reasons to perform this study in section 2. The goal of this study and the reachability criterion for the goal are expressed in section 3. In section 4, we represent our technique for functional requirements elicitation. This technique has been elaborated during a series of industrial projects on software requirements elicitation. We briefly mention these projects in section 5. We overview investigations related to our study and show the novelty of our study in section 6. We explain why we reach the goal of this study with our technique and come to some conclusions in section 7. Section 8 contains a reference list.

2. Motivation

To our mind, challenges associated with elicitation of functional requirements from the API documentation may be overcome by applying:

- a proper requirements elicitation process;
- effective solutions for technical and scientific problems;
- automation of labor intensive tasks.

We address all these aspects in this paper. We present a requirements elicitation process elaborated during a number of industrial projects. We also offer solutions to some markup issues of API documentation text. We automate routine and labor intensive tasks with our requirements management solution Requality [4].

Industrial requirements management tools [5] like Requality usually come along with a requirements management process [6]. These processes do not usually address the problem of requirements elicitation from written sources. We cover this gap with this paper presenting a requirements elicitation process for API documentation and similar documents like API-related standards.

3. Problem statement

Suppose we have an API documentation. One should extract functional requirements from this documentation, build a catalog of functional requirements and supplement the catalog with new requirements obtained from other sources.

A proper requirements elicitation technique should meet the following requirements:

1) The technique should construct a functional requirements catalog in which every requirement has the unique identifier.

2) The technique should support traceability of requirements to text fragments that represent those requirements in the API documentation.

   Such a traceability relation can be used to estimate coverage of the documentation text by the requirements markup.

3) The technique should tolerate possible changes in the API documentation text.

   Problems are often discovered in the documentation during requirements analysis. The author of the documentation fixes them and issues a corrected version of the documentation text. Some text fragments may be changed due to fixes. These changed text fragments might already be traced to existing requirements in the previous version of the documentation. In this case, we should transfer the existing mapping of requirements to text fragments onto the next (fixed) version of the documentation.

4) The technique should assist in requirements refinement which is the primary way to improve understanding of the system under test.
5) The technique should assist in building a complete set of requirements for functions, subsystems and the whole system.
6) The technique should not rely on a particular natural language.

4. Requirements elicitation technique

In this section, we describe different aspects of our technique for functional requirements elicitation.

4.1 Demo example

All examples in this paper refer to function *select* from POSIX [7] standard. This function waits until an event is registered for one of the file descriptors provided. There are three types of events:

- a file descriptor is ready for reading;
- a file descriptor is ready for writing;
- an error is registered for a file descriptor.

Function *select* is a complex one. The main reason is that the function handles different file types differently. There are three main file types:

- regular files;
- sockets;
- terminals and pseudo terminals.

In this paper, we assume for simplicity that *select* function is applied to regular files only.

4.2. Requirements catalog structure

Let us build our requirements catalog in the form of a *tree* [8] in which:

- vertices are requirements;
- an arc between two incident requirements represents the refinement relation, i.e. the requirement having the higher depth refines an aspect of the requirement having the lower depth.

Before inserting a new requirement into the requirements catalog the following characteristics should be checked [3]:

- the new requirement is interpreted unambiguously;
- the new requirement does not contradict the existing requirements;
- the new requirement cannot be logically deduced from the existing requirements;
- the new requirement has already been implemented or is going to be implemented and meets the actual system or user need;
- the new requirement defines a single aspect of a function, subsystem or system.

Proving each characteristic is a challenge but it is often not necessary. For instance, a requirement can be considered unambiguous if all team members interpret the requirement in the same way. A meeting [9] of all team members can be appointed to address this issue.

If one of the above conditions is not met, we should correct the new requirement or the requirements catalog or both. Thus, by adding a new requirement to the requirements catalog we either:

- refine another requirement;
- or improve completeness of requirements.

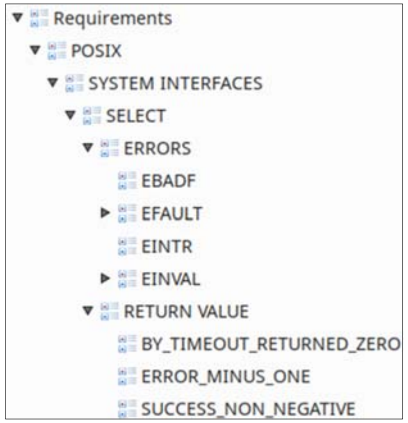

Fig. 1. A fragment of a requirements catalog

The requirement analyst assigns mnemonic names to the requirements while Requality requirements management tool automatically generates unique identifiers for them. The path from the root of the requirements catalog to a requirement may be used as a *unique link* to the requirement. Such a link is often used to trace a test to the requirement verified by the test. For instance, the unique link to requirement *EBADF* on fig. 1 is the following: */Requirements/SELECT/ERRORS/EBADF*.

The structure of the requirements catalog usually reproduces the structure of the API documentation up to a certain level. For instance:

1) The root requirement formulates the main task, goal or mission of the computer program.
2) The root requirements for the subsystems or main components of the computer program reside on the first level of the requirements tree (catalog) and formulate the main task or goal of the corresponding subsystem or component.
3) The root requirements for the classes and global functions (C programming language) reside on the second level and formulate the main task or goal of the corresponding class or function.
4) The root requirements for the methods of classes reside on the third level. Functional requirements for the global functions start to appear on third level too.
5) Functional requirements for the class methods begin on the fourth level.

Description of individual functions in the API documentation is often generated from structured source code comments, e.g. written with Javadoc, Doxygen and etc. Structured nature of source code comments leads to some structure in the description of functions in the API documentation. The structural description of functions in documentation, in turn, is projected onto the requirements catalog. For instance, description of *select* function in POSIX [7] contains the following structure of headings: *NAME, SYNOPSIS, DESCRIPTION, RETURN VALUE, ERRORS*. Sections *NAME* and *SYNOPSIS* do not contain any functional requirements. Sections *RETURN VALUE* and *ERRORS* describe the return value of the function and possible error codes respectively. The corresponding headings become child requirements of requirement *SELECT*.

4.3. Requirements text structure

Each requirement should have a textual description:

- a non-empty set of text fragments from the documentation;
- or a manually written text by a template.

Text fragment is a continuous word sequence in a document. Using Requality one can mark text fragments in a document and then link them to a requirement. It is possible to switch between text fragments and corresponding requirements with a single mouse click. There is, for example, the following text fragment in the description of function *select*: *if function select ends by time limit then return 0*. We assign this text fragment to requirement BY_TIMEOUT_RETURNED_ZERO. An additional manual description of the requirement is not mandatory because the text fragment describes the requirement properly.

Requirements have built-in attributes in Requality, e.g. a mnemonic name, a manual description, as well as user defined attributes. If needed, we write manual descriptions for functional requirements according to the following template [10]:

The function/subsystem/system MUST [actions set], [if/while/until CONDITION]. Actions can be:

- a modification of the internal state of the computer program;
- or return of a specific value from the function;
- or an exception.

Conditions are usually made up of logical predicates and logical operations like conjunction, disjunction and negation.

Here is an example. A manual description for requirement ERRORS can be as follows: *Function select MUST write the unique error code into variable [errno] if an error happened.*

A condition is skipped if actions are unconditional. If the subject of the actions is obvious then the phrase *Function/subsystem/system* MUST may also be omitted.

All requirements without text fragments from the API documentation (but with manual descriptions) are considered to come from other sources like an interview with developers or completeness analysis of requirements.

It is sometimes better to represent requirements in the form of tables, images and models. Requality supports tables and arbitrary images in manual descriptions of requirements.

4.4. Heuristic technique

We formulate a set of heuristics in this section that make a requirements elicitation technique effective. We tried to implement them in our technique.

1) Attempts to achieve the best possible requirements quality characteristics are often irrational. The quality of requirements should be enough to achieve the planned testing quality defined in the completeness criterion.
2) The API documentation is not supposed to contain the complete set of functional requirements. Therefore, the authors of the documentation, designers, developers and test engineers become an important source of functional requirements. They help to elicit and improve the requirements catalog [11].
3) The use of special requirements management tools like Requality greatly facilitates and simplifies the requirements elicitation process and the maintenance of large requirements catalogs.
4) Improvement of the API documentation is required to improve the quality of the requirements extracted from that documentation.
5) Representation of some requirements in a more suitable form rather than textual one and visual modeling of unclear requirements help to improve the requirements quality [7]. Effective non-textual requirements representations include tables and formulas. Data flow diagrams [12],

UML action diagrams, UML state diagrams, decision tables [13] and other models [14] can be used to model different aspects of requirements.

6) A group work on a complex problem like requirements elicitation is efficient since it implies mutual assistance and participation of engineers having complementary qualifications. However, an excessive team may have a negative impact on the efficiency of requirements elicitation.
7) The use of a task management system, bug tracking system and a version control system helps to support controllable, goal-oriented, responsible interaction of team members and to meet project deadlines.

4.5. Roles of participants

The process of requirements elicitation is a process of organized and controlled interaction of participants:

1) Requirement analyst (analyst):
   - helps technical project manager to assign priorities to functions;
   - extracts functional requirements from the API documentation;
   - supplements the requirements catalog with requirements from other sources;
   - reveals issues in the requirements catalog and fixes them;
   - fixes issues in the requirements catalog reported by other participants;
   - reveals issues in the API documentation;
   - participates in requirements verification procedures.
2) Author of the API documentation:
   - creates the API documentation;
   - reveals issues in the API documentation;
   - fixes issues in the API documentation;
   - as an important source of functional requirements provides them to the analyst;
   - participates in requirements verification procedures.
3) Test engineer:
   - formulates a test completeness criterion;
   - traces tests to requirements;
   - as an important source of functional requirements provides them to the analyst;
   - reports issues found in the requirements catalog during testing;
   - helps technical project manager to assign priorities to functions;
   - participates in requirements verification procedures.
4) Developer of the computer program (developer):
   - as an important source of functional requirements provides them to the analyst;
   - answers to the questions about the implementation of the computer program;
   - helps technical project manager to assign priorities to functions;
   - participates in requirements verification procedures.
5) Technical project manager (manager):
   - ensures the requirements elicitation process to meet its deadline;
   - assigns priorities to functions;
   - organizes productive interaction of all team members;

   - helps other participants to solve various issues they find it difficult to solve by oneself.

All participants use a version control system to track changes in the API documentation and requirements catalog as well as to assign release versions to them. All issues discovered in the API documentation and requirements catalog are trucked in a bug tracking system. We track all tasks (as opposed to issues) in a task management system. Typical tasks are the following: markup requirements for a function, verify completeness of the set of requirements for a function, assign priorities to functions.

4.6. Requirements elicitation process

4.6.1 Preparatory stage

At first, the analyst structures functions provided for testing by dividing them into subsets. A subset may contain interface functions to a single subsystem or functions performing related operations like send and receive, write and read.

Then the analyst looks up the pages in the API documentation describing each subset of functions generally and each function individually in details, i.e. the analyst constructs the appropriate relations between subsets of functions and documentation pages and between individual functions and documentation pages. The analyst supplements this relation during the whole requirements elicitation process.

Then the manager, the analyst and the test engineer assign priorities to the subsets of functions and to the individual functions as well. Priorities depend on different factors: relative complexity of functions, the size of the describing documentation text, restrictions imposed by the terms of reference document, available human and time resources, etc. Priorities are not static and are subject to change during the requirements elicitation process.

The participants of the requirements elicitation process track the progress of all activities using a task management system or a bug tracking system. They assign priorities to the tasks taking into account the priorities of the interface functions that are being worked on.

4.6.2 Requirements markup

The analyst marks up requirements in the documentation respecting the priorities of the interface functions. This process is accompanied by a number of problems.

Does the fragment of text describe a requirement? One can focus on the keywords that may indicate the presence of a requirement in a sentence [15]. For example, POSIX [7] standard requires to use the verb **must** in requirement statements: *Upon successful completion of the function, pselect () and select () must return the total number of bits specified in bitmasks.*

In general, a requirement cannot be recognized in a given set of text fragments on the basis of syntax rules. A functional requirement is a statement about what the computer program, a subsystem or a function should do under a condition or unconditionally. The object of actions in a software system is data, e.g. a return value of a function, the internal state of the computer program, an exception, a message. Let us assume that a set of text fragments formulates one or more requirements if:

- an action is performed or a number of actions;
- the subject of the action is the computer program, a subsystem or a function;
- the object of the action is some data;
- the action should be performed under a condition or unconditionally.

What should be done with text fragments that do not contain any requirements? Requality marks text fragments containing requirements (assigned to a requirement node) with a color. Unmarked text fragments have not been analyzed yet. To be able to control the completeness of documentation markup, text fragments that have been analyzed and are known to not contain any requirements should be separated from unmarked text fragments. Requality tool supports nodes of type *text node* (as opposed to *requirement* nodes). All text fragments that do not contain any requirements are assigned to nodes of this kind and are marked with a color.

How should the analyst resolve a text problem found in the API documentation? Some examples of text problems are the following: unclear meaning of a certain text fragment, a contradictory statement, an ambiguous statement.

The analyst should create a task in the bug tracking system and make the author of the API documentation responsible for it to be resolved.

Should the analyst refine context dependent text fragments? The context of text fragments can affect their meaning. We recommend to clarify context-dependent text fragments in the requirements catalog. Requality provides *description* attribute in requirement nodes for this purpose.

4.6.3 Suspension criterion for the requirements markup of a function

The requirements markup process for a function is gradually approaching a state when further progress is either limited or difficult. The main natural reason for this is the limited function description in the API documentation. In addition, a large number of issues may slow down the requirements markup process for a function. An iterative markup of functions has proven to be productive.

Criterion 4.1 (Suspension criterion for the requirements markup of a function). *For every text fragment in the function description the following should hold:*

- either the text fragment is marked up, i.e. referred to a requirement or a text node;
- or a task has been created in a bug tracking system concerning the text fragment.

The criterion can examine extra values if needed:

- the number of issues revealed in the function description;
- the amount of marked-up text;
- the amount of time spent on elicitation of requirements for the function;
- the complexity of the function expressed numerically.

4.6.4 Transfer of requirements catalog onto a new API documentation release

While the requirements catalog is being built over a certain documentation release, the API documentation is being naturally improved. When a new documentation release comes out, the requirements analyst should reflect the new improvements in the requirements.

Requality helps to transfer the existing requirements catalog onto a new documentation release. Text fragments that have not changed are mapped to the requirements catalog automatically. Other text fragments are mapped in a semiautomatic manner.

4.6.5 Verification of requirements catalog for a function

When the function description in the documentation has been completely marked up, it is necessary to decide whether the requirements are ready for testing or should be preliminary verified.

High complexity of a function is the key factor in favor of verification. However, complexity is difficult to estimate. The following values can be taken into account:

- the time spent to build the requirements catalog for the function;
- the number of issues in a bug tracking system related to the function;
- the size of the function description in the API documentation;
- the number of completed iterations of the requirements elicitation and etc.

Many existing verification techniques demonstrate effectiveness when applied to requirements: formal inspection [16], equivalence partitioning [17], boundary value analysis [18], decision table analysis [13], meeting [9], consultation, interview [19] and questionnaires [19] [20].

All issues found during verification should be tracked through the bug tracking system. We recommend to verify requirements according to a preliminary elaborated plan which may include the following information:

- the object of verification, i.e. the requirements for a function or a set of functions;
- the place, the date, the start and the finish times;
- a list of participants and their tasks;
- the subject of verification, i.e. the requirements properties to be verified such as uniqueness, completeness, consistency;
- verification method.

4.6.6 Necessary conditions to stop the requirements elicitation process for a function

The requirements analyst, software tester and developer should come to the same understanding of the functions' behavior. This can be achieved when they successfully complete all tasks dedicated to the function.

Proposition 4.1 (Necessary conditions to stop the requirements elicitation process for a function):

- *the markup of the function description in the API documentation should have been completed;*
- *all tasks in the bug tracking system directed to correction of the function description in the API documentation should have been completed;*
- *all tasks in the bug tracking system directed to correction of the requirements catalog for the function should have been completed;*
- *all tasks in the task management system directed to supplementation of the requirements catalog for the function from other sources should have been completed;*
- *all requirements verification tasks for the function in the task management system should have been completed and the subject of the verification included:*
    - *unambiguity of the requirements;*
    - *the accuracy of the leaf requirements in the requirements catalog;*
    - *consistency of the set of requirements;*
    - *completeness of the set of requirements.*
- *the function has been tested and all found errors have been fixed.*

4.6.7 Feedback from functional testing

API testing is always automated. Tests should be written in accordance with the functional requirements. Tracing establishes links between tests and requirements being verified. Thus, testing allows us to naturally confirm verify-ability of requirements.

The test engineer usually discovers many issues in the requirements during the requirements-based test design. In addition, some inconsistencies between the requirements and the observed program's behavior may be due to issues in the requirements. The test engineer should create a task in the bug tracking system for all discovered issues and make the requirements analyst responsible to resolve requirements' related issues.

Test development and analysis of the discovered issues highly improve the test engineer's understanding of the function's behavior. Therefore, the analyst should engage the test engineer in requirements verification and supplementation procedures. This will help to ensure the completeness of the requirements catalog and improve the overall quality of the requirements.

4.6.8 Necessary conditions to stop the requirements elicitation process

The whole requirements elicitation process finishes when all participants have successfully completed their tasks.

Proposition 4.2 (Necessary conditions to stop the requirements elicitation process):

- *the markup of the API documentation or its target part should have been completed;*
- *all tasks in the bug tracking system directed to correction of the API documentation should have been completed;*
- *all tasks in the bug tracking system directed to correction of the requirements catalog should have been completed;*
- *all tasks in the task management system directed to supplementation of the requirements catalog from other sources should have been completed;*
- *all requirements verification tasks in the task management system should have been completed;*
- *all target API functions should have been tested and all found errors have been fixed.*

5. Approbation

Our technique for functional requirements elicitation has evolved during a number of industrial projects. Those projects have been performed by the authors of this study and other researchers from the software engineering department of ISPRAS [21].

The technique has recently been used for requirements elicitation on input-output multiplexing functions from POSIX [7] standard:

- poll;
- select;
- pselect.

The above functions were implemented in a real time operating system. The API of the operating system was described in an API documentation. A requirements catalog consisting of 317 functional requirements has been built as a result of a multi-iterative requirements elicitation process. Dozens of errors were found in the API documentation including:

- incompleteness of information;
- inaccuracy of statements;
- conflicts with POSIX [7] standard.

We have also used Requality to build requirement catalogs for some parts of the following standards and specifications:

- ARINC 653 [22];
- TTCN-3 interface specifications;
- several RFCs including RFC 826, RFC 760 and RFC 768.

6. Related work

One way or another, individual ideas or solutions expressed in this paper might already be published in books or applied in practice. However, we couldn't find any requirements elicitation techniques characterized as ours:

- the method is aimed at a specific type of documentation, i.e. the user API documentation;
- the method uses feedback from functional testing to enhance the quality of requirements;
- the method is effective in practice due to the use of a specialized software tools like Requality requirements management tool.

Our research group developed requirements management tool Requality. We know for sure there are no publications concerning techniques for requirements elicitation on the basis of this software tool. We fill this gap by publishing this study.

There are several alternative industrial tools [5]. Most of them are used in requirements development from scratch. They assist in building requirements catalogs and support relations between requirements originated at different levels of the software life cycle:

- business requirements;
- system requirements;
- functional requirements.

But most of these requirements management tools cannot maintain links between documentation fragments and related requirements as Requality does.

NLP (natural language processing) methods [23] are widely used to extract various information like requirements from texts written in a natural language. NLP methods are usually well automated therefore they effectively analyze big text data.

Information about hierarchy of classes and methods of these classes is extracted with NLP methods from the API documentation in study [24]. Methods are divided into categories:

- create a resource;
- lock access to a resource;
- modify a resource;
- unlock access to a resource;
- delete a resource.

Then an automaton is created for every resource on the basis of the extracted information. The automaton is then used to reveal defects in the computer program. For instance, the use of a resource before creating it.

Our study and work [24] are similar in that we analyze the same type of documentation, i.e. the API documentation. The both studies have the same goal to improve the quality of computer programs. However, the methods to reach this goal are different. We look for functional requirements. The authors of study [24] look for defects.

NLP methods can be used to verify some characteristics of requirements. For instance, software tool QuARS [25] can reveal ambiguity of text and subjectivity of text (not a requirement but a personal opinion). LOLITA [26] analyzes text and incorporates it in a semantic net [27]. Then possible text interpretations are looked up.

Complex lexis, syntax and morphology of some natural languages and many exceptions from the language rules make it more difficult to apply NLP methods to analyze texts written in those languages. From one hand, these complexities restrict application of NLP methods. From the other hand, these complexities simply become responsibilities of the analyst in our technique.

Application of several requirements elicitation techniques leads to synergistic effect. The choice of a complementary technique depends on human resources available, i.e. the number of engineers, their experience, qualification and etc. Techniques effectively complementing our requirements elicitation technique do not strictly relate to mark up of texts. Among them are the following methods:

- formal inspection [16];
- equivalence partitioning [17];
- boundary value analysis [18];
- decision table analysis [13];
- meeting [9];
- consultation;
- interview [19];
- questionnaires [19] [20].

Correction of problems in the API documentation is an important part of our requirements elicitation technique. There are methods specially designed to reveal errors in the API documentation. For instance, texts written in a natural language are analyzed with NLP methods and source code snippets are analyzed by a code analyzer in study [28]. Combination of two different types of analyzes helps to find inconsistencies between a text fragment and a source code snippet.

A method for requirements elicitation from the user documentation for a legacy system is proposed in study [29]. The extracted requirements are then used to create a functional specification document for a new system similar to the legacy system. Text structure, key words, lexical, syntactical and other text characteristics are used to extract key features of the system, functional requirements, use cases and nonfunctional requirements.

7. Conclusion

In this paper, we propose a technique for functional requirements elicitation from the user API documentation. By means of this technique the requirements analyst can create a catalog of functional requirements suitable for functional testing.

The requirements catalog is a tree in which every requirement has a unique identifier. This tree structure assists in functional requirements refinement and usually reproduces the structure of the API documentation up to a certain level.

Markup of all documentation text with requirements is a necessary condition for a requirements set to be complete. Our requirements management tool Requality maintains links between documentation fragments and related requirements helping us to transfer the requirements catalog onto upcoming documentation versions.

We support requirements obtained from non-written sources, e.g. provided by team members or as a result of a requirements analysis. We write down them in a natural language by a template and replenish the requirements catalog with them.

An acceptable quality of requirements is obtained due to systematic verification procedures, feedback from functional testing and team collaboration through a version control system, a bug tracking system and a task management system.

References / Список литературы

[1] DO-178C. Software Considerations in Airborne Systems and Equipment Certification. RTCA SC-205 and EUROCAE WG-12 Std., 01 2012.

[2] V.V. Kulyamin, N.V. Pakulin et al. Formalization of requirements in practice. Preprints of the Institute for System Programming of the Russian Academy of Sciences, Preprint 13, 2006, 70 p. (in Russian) / В.В. Кулямин, Н.В. Пакулин и др. Формализация требований на практике. Препринты Института системного программирования РАН, Препринт 13, 2006 г., 70 стр.

[3] ISO/IEC/IEEE International Standard - Systems and software engineering – Life cycle processes – Requirements engineering, ISO and IEC and IEEE Std., 11 2018.

[4] D. Kildishev and A. Khoroshilov. Developing requirements management tool for safety-critical systems. In Proceedings of the International Conference on Actual Problems of Systems and Software Engineering, 2019, pp. 50-57.

[5] N. Gorelits, D. Kildishev, and A. Khoroshilov. Requirement management for safety-critical systems. Overview of solutions. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 1, 2019. pp. 25-48 (in Russian). DOI: 10.15514/ISPRAS-2019-31(1)-2 / Н.К. Горелиц, Д.С Кильдишев, А.В. Хорошилов. Управление требованиями к ответственным системам. Обзор решений. Труды ИСП РАН, том 31, вып. 1, 2019 г., стр. 25-48.

[6] P. Zielczynski. Requirements Management Using IBM Rational RequisitePro. IBM Press, 2007, 360 p.

[7] Portable Operating System Interface, ISO and IEC and JTC 1/SC 22 Std., Rev. 9945:2009, 09 2009.

[8] A. Khoroshilov and D. Kildishev. Formalizing metamodel of requirements management system. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 5, 2018, pp. 163-176. DOI: 10.15514/ISPRAS-2018-30(5)-10.

[9] R. Ocker, J. Fjermestad et al. Effects of four modes of group communication on the outcomes of software requirements determination. Journal of Management Information Systems, vol. 15, issue 6, 1998, pp. 99-118.

[10] K. Wiegers and J. Beatty. Software Requirements. Microsoft Press, 2013, 672 p.

[11] Z. Zhang. Effective requirements development – A comparison of requirements elicitation techniques. In Proc. of the International Conference on Software Quality Management, 2007, pp. 225–240.

[12] T. DeMarco. Structure Analysis and System Specification. In Pioneers and Their Contributions to Software Engineering, Springer, 1979, pp. 255-288.

[13] R. Shiffman and R. Greenes. Improving clinical guidelines with logic and decision-table techniques: Application to hepatitis immunization recommendations. Medical Decision Making, vol. 14, no. 3, 1994, pp. 245-254.

[14] J. Beatty and A. Chen. Visual models for software requirements. Microsoft Press, 2012, 480 p.

[15] N. Niu and S. Easterbrook. Extracting and modeling product line functional requirements. In Proc. of the 16th IEEE International Requirements Engineering Conference, 2008, pp. 155-164.

[16] M. Fagan. Design and code inspections to reduce errors in program development. IBM Systems Journal, vol. 15, no. 3, 1976, pp. 182-211

[17] D. Richardson and L. Clarke. A partition analysis method to increase program reliability. In Proc. of the 5th International Conference on Software Engineering, 1981, pp. 244-253.

[18] S. Reid. An empirical analysis of equivalence partitioning, boundary value analysis and random testing. in Proc. of the Fourth International Software Metrics Symposium, 1997, pp. 64-73.

[19] S. Sharma and S. Pandey. Article: Revisiting requirements elicitation techniques. International Journal of Computer Applications, vol. 75, no. 12, 2013, pp. 35-39.

[20] S. Kimani, E. Panizzi et al. Digital Library Requirements: A Questionnaire-Based Study. In Handbook of Research on Digital Libraries: Design, Development, and Impact. Information Science Reference, 2009, pp. 287-297.

[21] Open-Source Projects. Available: https://forge.ispras.ru/projects

[22] S. Santos, J. Rufino et al. A portable ARINC 653 standard interface. In Proc. of the 2008 IEEE/AIAA 27th Digital Avionics Systems Conference, 2008, pp. 1.E.2-1-1.E.2-7.

[23] J. Eisenstein. Introduction to Natural Language Processing. MIT Press, 2019, 536 p.

[24] H. Zhong, L. Zhang et al. Inferring specifications for resources from natural language api documentation. In Proc. of the 2009 IEEE/ACM International Conference on Automated Software Engineering, 2009, pp. 307-318.

[25] G. Lami. Quars: A tool for analyzing requirement. Technical report CMU/SEI-2005-TR-014 ESC-TR-2005-014, Carnegie Mellon University, Software Engineering Institute, 2005.

[26] L. Mich. Nl-oops: From natural language to object oriented requirements using the natural language processing system Lolita. Natural Language Engineering, vol. 2, no. 2, 1996, pp. 161-187.

[27] L. Schubert, R. Goebel, and N. Cercone. The structure and organization of a semantic net for comprehension and inference. In Associative Networks. Academic Press, 1979, pp. 121-175.

[28] H. Zhong and Z. Su. Detecting api documentation errors. In Proc. of the 2013 ACM SIGPLAN International Conference on Object-Oriented Programming Systems, Languages and Applications, 2013, pp. 803-816.

[29] I. John and J. Dörr. Elicitation of requirements from user documentation. In Proc. of the Ninth International Workshop on Requirements Engineering: Foundation for Software Quality, 2003.

Information about authors / Информация об авторах

Evgeny Anatolyevich GERLITS – researcher at the Software Engineering Department of the Institute for System Programming of the RAS. Main research interests: software dynamic verification, software testing, software quality assurance, static and dynamic program analysis.

Евгений Анатольевич ГЕРЛИЦ – научный сотрудник Института системного программирования. Сфера научных интересов: методы динамической верификации программ, методы тестирования и обеспечения качества программного обеспечения, статический и динамический анализ программ.

Denis Stepanovich KILDISHEV is a junior researcher of the Software Engineering Department. His research interests include requirements management tool development.

Денис Степанович КИЛЬДИШЕВ является младшим научным сотрудником отдела технологий программирования. Его научные интересы включают разработку инструмента управления требованиями.

Alexey Vladimirovich KHOROSHILOV – Ph.D. in Physics and Mathematics, Leading Researcher, Director of the Linux OS Verification Center at ISP RAS, Associate Professor of System Programming Departments at Moscow State University, the Higher School of Economics, and Moscow Institute of Physics and Technology. Main research interests: design and development methods for critical systems, formal methods of software engineering, verification and validation methods, model-based testing, requirements analysis methods, Linux operating system.

Алексей Владимирович ХОРОШИЛОВ – кандидат физико-математических наук, ведущий научный сотрудник, директор Центра верификации ОС Linux в ИСП РАН, доцент кафедр системного программирования МГУ, ВШЭ и МФТИ. Основные научные интересы: методы проектирования и разработки ответственных систем, формальные методы программной инженерии, методы верификации и валидации, тестирование на основе моделей, методы анализа требований, операционная система Linux.