INFORMATICA, 2006, Vol. 17, No. 3, 325–346

Methodology to Evaluate the Functionality of Specification Languages

Jelena GASPEROVIC, Albertas CAPLINSKAS
Software Engineering Department, Institute of Mathematics and Informatics
Goštauto 12, LT-01108 Vilnius, Lithuania
e-mail: j.gasperovic@algoritmusistemos.lt , alcapl@ktl.mii.lt

Received: December 2005

Abstract. The paper proposes a methodology for evaluation of specification language functionality characteristics. It describes background of the proposed methodology, discusses the methodology in detail, and shortly describes experimental results obtained using the proposed methodology to evaluate the functionality of Z and UML languages.

Key words: specification languages, internal quality, quality characteristics, quality evaluation.

1. Introduction

It is possible to speak about internal quality and quality in use of a specification language. Internal quality is a descriptive characteristic that describes the quality of a language independently from any particular context of its use. Meanwhile, quality in use is evaluative characteristic of a language obtained by making a judgment based on criteria that determine the worthiness of a language for a particular project (Caplinskas and Gasperovic, 2005c). However, it is impossible to evaluate the quality in use without knowing characteristics of internal quality. Despite the long history of specification languages, their quality is almost unstudied. Particularly, it is very little known how to evaluate the functionality of a specification language. Several authors (Jackson, 1999; Wand and Weber, 1989; Wand and Weber, 1995; Opdahl, 1997; Milton *et al.*, 1998; Mylopolous, 1998; Lindland *et al.*, 1994; Krogstie, 2003) investigated the problem of specification language quality. However, their results are fragmental and not enough systematic. A sketch of the systematic specification languages quality theory have been proposed in (Caplinskas and Gasperovic, 2005a; Caplinskas and Gasperovic, 2005b; Caplinskas and Gasperovic, 2005c). However, the question how to evaluate elementary quality characteristics, including the characteristics of the functionality, has also left uninvestigated in these works. For several reasons, it is a hard problem. Firstly, only several elementary characteristics can be measured directly. Secondly, internal quality is relative. Elementary characteristics of a specification language have different weights. Although the internal quality does not depend on any particular context, the global context should be taken into account. It means that the weights of characteristics (their importance) depend on the properties of the current population of Software Systems. Some characteristics are necessary for most of the

population, while the others are used very rarely. Thus, when the structure of population of Software Systems changes, the degree of languages quality changes too. Because it is impossible to investigate all current population of Software Systems, the weights of characteristics have probabilistic character. Finally, the evaluation results should be assessed and interpreted in a correct way. All three mentioned issues are complex and intricate.

The main purpose of this paper is to propose a methodology to evaluate the elementary characteristics of the functionality of a specification language. Although many components of the proposed methodology can be applied to evaluate elementary characteristics of reliability, efficiency, and usability of a specification language, for each of the mentioned cases the methodology as a whole should be elaborated and adapted separately.

Shortly, the proposed approach can be described as follows. All theoretically possible Software Systems are divided into categories of systems that meet principally different requirements. Then from the current population of Software Systems a sample for each category of systems should be chosen via purposive judgment sampling. After this using domain analysis methods for each sample feature model should be developed. Further these models should be refined and a number of Software Systems requirements specifications should be produced. These specifications are used as evaluation test examples to evaluate different specification languages. In order to do this, a set of evaluation tests and test suites should be developed and quality evaluation plan should be prepared. Finally, the obtained results are interpreted using the proposed procedure and the values of elementary characteristics are calculated.

The rest of the paper is organised as follows. Section 2 discusses the details of specification language quality evaluation problem. Section 3 proposes the methodology to evaluate elementary characteristics of the functionality of a specification language and describes its components in details. Finally, Section 4 concludes the paper.

2. The Main Specification Languages Quality Evaluation Problems

This paper proposes a methodology to evaluate the characteristics of internal quality of a specification language. Let us consider shortly the whole specification languages quality evaluation problem.

According to (ISO/IEC 14598-1:1999, 1999) "*quality evaluation is a systematic examination of the extent to which an entity (part, product, service or organisation) is capable of meeting specified requirements*". As pointed out in (Caplinskas and Gasperovic, 2005c), quality of a specification language may be evaluated both independently from any context of use and by making a judgment based on criteria that determine the usefulness of a language for a particular project. The first one is called internal quality. It is the quality of a specification language itself. The second one is called quality in use. It describes the extent to which a language used by specified users meets their needs to achieve specified goals in specified context of use. So, internal quality is descriptive and quality in use is evaluative characteristic of a specification language.

The main quality evaluation scheme is presented in Fig. 1.

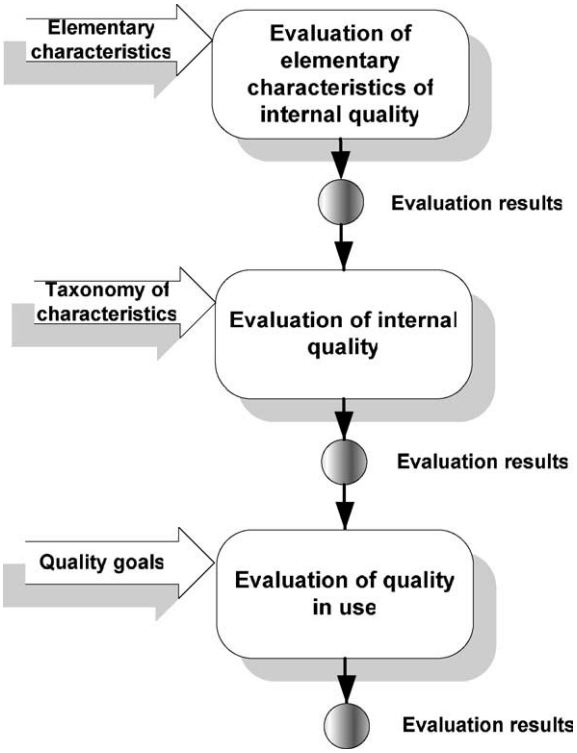

Fig. 1. The main quality evaluation scheme.

Internal quality is described by a set of quality characteristics. Usually a set of characteristics forms some hierarchical taxonomy. It means that, evaluating internal quality, different groups of related elementary characteristics are aggregated into characteristics of higher level and this process is repeated until the quality is described at the higher level by single aggregated characteristic. The theory for evaluation of internal quality of specification languages is still at the very beginning. Neither exhaustive commonly accepted set of elementary characteristics nor propositions how to measure and classify these characteristics have been described in scientific literature. The set of elementary characteristics to describe internal quality of a specification language and their taxonomy have been proposed in (Caplinskas and Gasperovic, 2005a).

This proposal has been based on the careful conceptual analysis of wide spectrum of specification languages, including UML, Z, VDL, Troll, and Alloy, as well as on the analysis of quality characteristics that are used to describe quality of similar artefacts such as programming languages and conceptual models. The proposed set of elementary characteristics includes 34 elementary characteristics. The proposed taxonomy is based on the classification scheme that is similar to the one described by ISO/IEC 9126 standard (ISO/IEC 9126, 1991). This taxonomy has five levels and provides that the quality of a specification language can be characterised by its functionality, reliability, usability, and efficiency. However, in (Caplinskas and Gasperovic, 2005a) these characteristics are

decomposed further in the different way than in ISO/IEC 9126.

In (Caplinskas and Gasperovic, 2005b) it has been proposed to express the values of characteristics of internal quality as probabilities that corresponding feature of a language **L** will be sufficient to specify relevant requirements of any theoretically possible Software System. It means that internal quality of specification language **L** is described as a probability that this language will satisfy the needs of any possible Software System development project regardless of its purpose, complexity, size and other specifics. Methods for aggregation of the values of characteristics of internal quality have been proposed in (Caplinskas and Gasperovic, 2005c).

Using proposed methods the values of elementary characteristics can be aggregated up to a single value, describing the internal quality of the whole language **L**.

In (Caplinskas and Gasperovic, 2005a) it has also been proposed how to evaluate the quality in use of the specification language **L** on the basis of its internal quality. Quality in use is evaluated for a given Software System development project **P** and describes the degree of appropriateness of the language **L** for the project **P**. In order to evaluate quality in use, the quality goals of the project **P** must be specified.

So, coming back to Fig. 1, the only missing point is the evaluation of elementary characteristics of internal quality. This paper is devoted to the issues of evaluation of elementary characteristics of the functionality of a given specification language **L**.

3. Evaluation of Elementary Characteristics of the Functionality

3.1. *Questions to be Answered*

Elementary characteristics of internal quality are the lowest-level characteristics. Thus, we should evaluate these characteristics in some way. Best of all would be to measure elementary characteristics, but it is possible for some characteristics only. In general, the evaluation techniques depend on the kind of characteristics or, in other words, depend on the aspect of the language (functionality, reliability, efficiency or usability), which these characteristics should describe.

In this paper we discuss only those techniques that are related to evaluation of characteristics describing the functionality of the language **L**. However, it is not known how to measure these characteristics directly.

In the proposed approach values of elementary characteristics describe probabilities that corresponding feature of a language **L** will be sufficient to specify relevant requirements of any theoretically possible Software System. However, it is impossible to calculate these probabilities directly, because it is impossible to have sufficient statistics about requirements of all theoretically possible Software Systems or even of the entire current population of Software Systems. So we propose to combine non-statistical sampling, domain engineering, and testing theory methods for this aim. The main idea is to develop a library of representative examples for each category of Software Systems and to use these examples as evaluation test examples to test the sufficiency of a specification language **L**

for specification of requirements of any systems from the current population of Software Systems. We propose to use a multi-stage sampling scheme, which provides that all theoretically possible Software Systems should be first divided into categories of systems that meet principally different requirements, then from the population of really existing systems a sample for each category of Software Systems should be chosen via purposive judgment sampling. After this using domain analysis methods for each sample feature model should be developed. Further these models should be used to develop a number of representative examples (evaluation test examples) to test functionality, reliability, efficiency and usability of specification languages.

To make this approach practically usable, we must answer several significant questions:

- In which way to classify Software Systems, that is what taxonomy of software Systems is better to choose for this aim?
- What information should contain an evaluation test example and how to choose sample and collect data that is necessary for development of such examples? How to evaluate relevance, reliability, and validity of the sample? What are the requirements for the collected data?
- How to analyse and interpret sampling results?
- In which way to describe and represent evaluation test examples in the library?
- How many evaluation test examples are needed to evaluate internal quality of a specification language?
- What should be the structure of evaluation test example? How to construct the suites of evaluation test examples? What kind of testing methodology to apply? How to evaluate the test coverage?
- In which way to describe quality evaluation results and how to calculate probabilities that corresponding features of a language **L** will be sufficient to specify relevant requirements of any theoretically possible system that belongs to currently known categories of Software Systems?

The remaining part of this section aims to answer these questions.

3.2. *Taxonomy of Software Systems*

Many different taxonomies of systems have been proposed in the literature. For example, Information Systems were classified into strategic-level, management-level, knowledge-level and operational-level systems (O'Brien, 2000). The kind of taxonomy depends on its intended use. In our case the classification should be done in such a way that each class of the systems should be characterised by some group of requirements specific for this class only. Additionally, the subject of our considerations is rather software constituent of Information Systems, because only properties of software should be specified using a specification language **L**. Our aim is to evaluate functionality of the language **L**, which is defined as the set of language features intended to be used for specification of requirements (Caplinskas and Gasperovic, 2005b). Taxonomy relevant to our aims has been proposed by Michael Jackson (Jackson, 1995; Jackson, 2001).

Jackson suggests that the main classification criteria of Software Systems should be the kind of problems solvable by the system. He proposed problem-oriented methodology – problem frames – that is purposed to characterise systems of different classes. Key concepts of this methodology are world, phenomena, domains, and descriptions. Problems are all located in the world. World phenomenology includes entities, events, values, states, truths, and roles. A domain is a distinct part of the world or, in other words, a collection of related phenomena. Phenomena and their relationships constitute domain properties. Domains can share phenomena. The only way two domains can interact is by an interface of shared phenomena (Jackson, 2000). System (*machine* in Jackson terms) is seen as a separate domain. Generally, there are four kinds of domains: systems (*machines* in Jackson terms), causal domains, lexical domains, and biddable domains. Essential properties of a causal domain are causal relationships. These relationships allow the system to cause and constrain events and state changes in the domain. The significance of the lexical domain is in the values and the truths and other relationships among them. The relationships here are not causal but definitional. Biddable domains represent users and operators, because their correct behaviour is described in instructions and they are bidden to follow the instructions.

Properties of problem domain describe facts about real world that are entirely beyond the control or influence of the system that should be built (Bray, 2002). These are knowledge that is required to solve the problem and, in parallel to requirements, should be described using specification language. Problem frames model the problem domain as a set of inter-related subdomains, where a subdomain is any part of the problem domain that may be usefully singled out (Bray, 2002). The effects that the system should produce also should be described using specification language. Jackson terms this description as requirements. Problem frames model requirements, too. The system interacts with the real world through the interface of shared phenomena. Typically, shared phenomena are events and states. Problem frames are represented using special graphical notation. Such representation is called Problem Diagram (Fig. 2).

Jackson identified seven elementary categories of the systems:

- **Transformation systems** – where the system must transform input data in a particular format into output data in a corresponding particular format.
- **Control systems** – where the system must control the behaviour of some part of the real world.
- **Commanded behaviour systems** – where the system must control the behaviour of some part of the real world in accordance with commands issued by an operator.

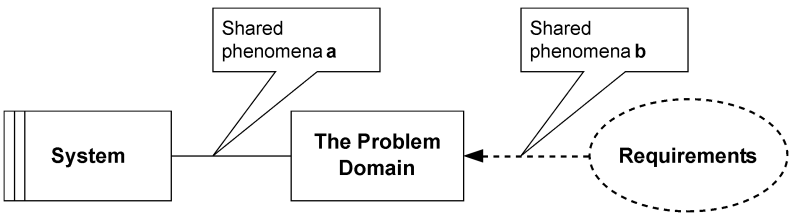

Fig. 2. Problem diagram.

- **Workpiece systems** – where the system must perform directed operations upon objects that exist only within the system.
- **Connection systems** – where the system must maintain correspondence between subdomains that are not directly connected.
- **Information display systems** – where the system must maintain a continuous display of information about an autonomous dynamic real world.
- **Information answer systems** – where the system must handle requests for information about the problem domain.

Whilst Jackson explicitly made no claim that he has identified all the elementary frames, as far as it is known, to date, only one additional frame – the simulator frame – has been proposed (Bray and Cox, 2003).

More complex systems can, generally, be described using composite frames composed of two or more interacting, elementary frames. To construct a multi-frame the system must be seen as composition of conceptual subsystems, each from which can be described by an elementary frame. Some elementary frames may partly overlap, for example, they may share some subdomains of problem domain. Of course, the problem should be partitioned into elementary subproblems, too. On the other hand, it is possible to distinguish subtypes of the systems described by elementary frames. For this aim elementary frames must be enriched using so called variants. A variant typically adds additional subdomain to the problem domain and supplementary requirements. Jackson proposes (Jackson, 2001) four kinds of variants: description, operator, connection, and control variants. Description variant introduces a description lexical domain, a biddable operator domain is included into operator variant, a connection variant provides a connection domain between the system and the problem domain, with which it interfaces, and a control variant introduces no new domain, but it changes the control characteristics of interface phenomena.

3.3. *Development of Evaluation Test Examples*

In statistics sample is defined as a part or subset of some population taken to be representative of this population as a whole for some investigative purposes of research (Cochran, 1977). In the context of this paper the term population refers to all systems of the particular category of Software Systems. However, any real system has features of several elementary categories of Software Systems. So, we suppose that our population consists of the systems, in which features of the particular elementary category are mandatory. For example, many real portals can be classified as content management systems or as "chatting room" or even as workflow management systems. However, any portal first of all is a content management system and we may consider all portals as a population, which represents some subcategory of information answer systems.

The technique of selecting a suitable sample is called sampling. There are two basic kinds of sampling: statistical (probability) and non-statistical (non-probability). Statistical sampling is also called random sampling. Non-statistical sampling is any sampling technique, in which the probability of a population element being chosen is unknown.

There are two basic kinds of non-probability sampling: accidental sampling and purposive sampling (Cochran, 1977). Patton defines purposive sampling as a "*sampling procedure that selects information rich cases for in-depth study*" (Patton, 1990). It means that during purposive sampling the sample is always intentionally selected according to the needs of the study. Judgment sampling is a kind of purposive sampling. It is a sampling technique in which special expertise is used to choose representative population elements. According to William M. Trochim

> "*The difference between non-probability and probability sampling is that non-probability sampling does not involve random selection and probability sampling does. Does that mean that non-probability samples aren't representative of the population? Not necessarily. But it does mean that non-probability samples cannot depend upon the rationale of probability theory. At least with a probabilistic sample, we know the odds or probability that we have represented the population well. We are able to estimate confidence intervals for the statistic. With non-probability samples, we may or may not represent the population well, and it will often be hard for us to know how well we've done so. In general, researchers prefer probabilistic or random sampling methods over non-probabilistic ones, and consider them to be more accurate and rigorous"* (Trochim, 2002).

So, evaluation of relative sampling risk is a hard problem for any non-statistical sampling approach, including purposive judgment sampling. However, although the reliability of the sample cannot be measured, there are other ways to ensure acceptable reliability. It can be done requiring that the judgement about which element of population should be included into sample should be made by an expert in the field and that this judgement should be made using three basic criteria: representativeness of the element, value of the element, and its relative sampling risk. According to the first criterion, the system is representative enough to be chosen, if it conforms to the appropriate problem frame. It means that the selected system must be information rich or, in other words, all the information that is provided by the appropriate problem frame should be necessary in order to develop this system. The main tool to support evaluation of the representativeness is sampling questionnaire that should be developed for each category of the systems. According to the second criterion, the system is valuable enough to be chosen, if it still is up-to-date; if it is popular enough among users, and if its intended application area is important enough. According to the third criterion, the relative sampling risk to choose the system is acceptable, if it is expected that the requirements of this system can be in some way elicited anew more or less completely. Such expectations can be recognised as justified in case, when the system requirements specification is available, or in case, when it is possible to contact the system developers, or in case, when the system provides at least exhaustive helps, demos, the system itself is available, and its non-functional requirements can be derived from some formal or informal standards of its intended use and its using modes. In addition, data quality requirements for collected questionnaire data should be stated in such a way that the relative sampling risk would be minimised. Data quality requirements define the required degree of data validity, reliability, consistency, accuracy, completeness, and the level of detail. To be more precise, data validity requirements define the extent to which questionnaire data should conform real features of the

system and its other properties, data reliability requirements define the degree to which questionnaire data should be free of errors, data consistency requirements define terms and classifications that should be used answering to questionnaire questions, data accuracy requirements define the degree to which the questionnaire data correctly describe the system, and data completeness requirements define the degree to which questionnaire data should be exhaustive. Before starting data analysis, the expert should obtain sufficient, competent, and relevant evidence that questionnaire data meet requirements. However, even for statistical sampling there is no effective statistical model for bringing together all these characteristics of quality into a single indicator. Additionally, except for very simple cases, there is no general statistical model for determining whether one particular set of quality characteristics provides higher overall quality than another. Thus, the only practical way to check that questionnaire data meets quality requirements is to apply manual data editing procedures. Another expert than the one who has collected data should edit data. He should detect unanswered questions, inaccurate answers, unrecorded answers and other violations of quality requirements.

Despite the difficulties in evaluation of the relative sampling risk, there are two reasons to prefer purposive judgement sampling against statistical sampling. Firstly, it is impossible to define sampling frame, because, in general case, the size of the population is unknown. Of course, for any category of Software Systems the population is large and finite, but it is impossible to say, even approximately, how large it is. Secondly, it is practically impossible by random sampling to take into account the value of the chosen system, which is very important for our purposes.

For analysis and interpretation of sampling results we propose to use feature-oriented domain analysis methods. Feature-oriented domain analysis usually is defined as

> "*the process of identifying, collecting, organising, and representing the relevant information in a domain, based upon the study of existing systems and their development histories, knowledge captured from domain experts, underlying theory, and emerging technology within a domain*" (Kang *et al.*, 1990).

The aim of domain analysis is to discover features of each system chosen for the sample and to produce feature-oriented model, which describes features of a generic system for examined category of Software Systems. Because almost all real systems are described using several problem frames, this generic system reflects the properties of several categories of Software Systems, too. However, only some features are mandatory for all sampled systems and it is legitimately to suppose that the generic system may be used as a basis to design evaluation test example for the category of Software Systems that is characterised by these features. Of course, this test is not necessarily exhaustive and additional tests likely will be necessary. On the other hand, this test may be used for additional testing of categories, which are characterised by some optional features.

There are several feature-oriented domain analysis techniques. The essence of each technique is increasing understanding of the analysed systems by capturing the information in formal models. In our approach, the Feature-Oriented Domain Analysis (FODA) technique (Kang *et al.*, 1990) is used. The feature-oriented model, developed using this technique, describes common and variable properties of all sampled systems and the dependences between these properties.

Thus, we propose the following methodology for the development of evaluation test examples for specification language:

1. **Framing**. Some category of Software Systems should be chosen and candidates for sampling should be identified, using criteria of representativeness, worthiness, and relative sampling risk.
2. **Development of vocabulary**. The different systems chosen for the sample may use different terminology and even be conceptualised in different ways. Thus, some conceptualisation should be chosen for the analysis, the basic terms used to model this system should be defined, and their equivalents in sampled systems should be listed. This vocabulary should describe concepts of chosen ontology and relations between these concepts. It is developed in a step-by-step manner, when analysis of the sampled systems progresses.
3. **Development of questionnaire**. The sampling questionnaire and data requirements should be developed. Both, questionnaire and data requirements, should be formulated in terms of vocabulary. Outside expert should validate completeness and correctness of the questionnaire and data requirements.
4. **Data collection**. Each sampled system should be analysed and all questions provided by the questionnaire should be answered. Decomposition should be used to support analysis. Each analysed multi-frame system should be decomposed into elementary frames and each elementary frame should be analysed autonomously.
5. **Data edition**. Inaccurate answers, unrecorded answers and other violations of quality requirements should be detected and corrected. Data edition should be done by some outside expert.
6. **Modelling**. Feature model should be developed for each sampled system. Modelling proceeds in bottom-up manner starting from the elementary frames.
7. **Synthesis**. Feature models of all sampled systems should be combined in order to produce the feature model of the generic system that is meant to be representative for the chosen category of Software Systems. The following rules should be applied to define features of the generic system:
   - Features, which are identical for all sampled systems, are defined as mandatory for the generic system.
   - Features, which differ for different subsets of sampled systems, however, can be generalised to one mandatory feature of the generic system, are defined as alternative subfeatures of this feature.
   - All other features of sampled systems are defined as optional for the generic system.
8. **Requirements elicitation**. Functional and non-functional requirements for the generic system should be derived from the feature model and documented (see Subsection 3.4). The requirements specification of the generic system should include requirements for all features provided by the feature model. Optional and alternative requirements should be annotated correspondingly. The main tool to support requirements derivation is refinement of features and parameterisation. Each feature should be refined into corresponding group of functional and non-functional

ID	Feature	Description	Rationale	Type	Composition rules
<Hierarchical number>	<Feature name>	<Text>	<Text>	[Optionall Alternative| Mandatory]	[Mutually exclusive with: <feature names>| Requires: <feature names>]

Fig. 3. Feature table.

requirements. The instances of entities mentioned in requirements should be replaced by corresponding variables. Composition rules (Fig. 3) should be included into the requirements specification.

We propose a tabular notation, feature tables (Fig. 3), which should be used to describe feature models. The main reason to use tabular representation is that it is hard to understand and to analyse large feature models represented using the original FODA graphical notation. In addition, the tabular notation allows collecting together all information about the feature that in the original FODA approach is distributed among feature diagrams, feature definitions and rationale of features.

Feature table (Fig. 3) represents FODA feature hierarchy. In this table feature's ID is a unique identifier of the feature, which reflects all previous levels of the feature hierarchy. The column "Type" describes the type of the feature. Common properties are addressed as mandatory or as alternative features and variable features are addressed as optional features. Composition rules define the semantics existing between features. They cannot be expressed in the feature diagram. There are two types of composition rules: mutual dependency (Requires) and mutual exclusion (Mutually exclusive with). All optional and alternative features that cannot be combined with the feature described in this row are listed after "Mutually exclusive with:" statement. All optional and alternative features that must be necessarily combined with this feature are listed after "Requires:" statement (Kang *et al.*, 1990).

Fig. 5 describes in the tabular form the fragment of the portal feature diagram presented in Fig. 4.

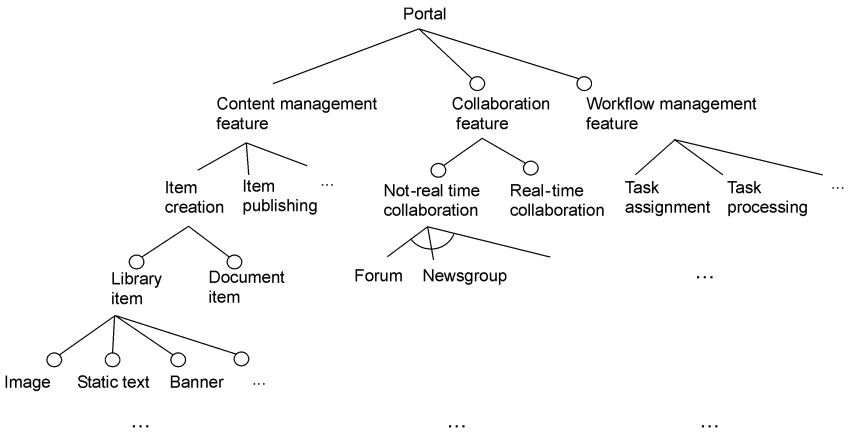

Fig. 4. Fragment of portal feature diagram.

Level	Feature	Description	Rationale	Type	Composition rules
1	**Content management feature**	**A service that allows creation, management, and publishing of portal content.**		**Mandatory**	
1.1	Item creation	A service that allows creation of content items.		Mandatory	
1.1.1	Library item	A service that allows creation of reusable, unstructured pieces of content.	If it is necessary to store parts of documents for re-use	Optional	
1.1.1.1	Image	A service that allows creation of a visual representation of an object, scene, person, abstraction, etc. produced on a Web page.	If images should be reused in many documents.	Optional	
…	…	…	…	…	…
1.2	Item publishing	A service that allows making content items accessible to portal users.		Mandatory	Requires: Item creation
2	**Collaboration feature**	**An online service that provides means of communication between portal users.**	**If communication between portal users is required.**	**Optional**	
2.1	Not real-time collaboration	A service that allows collaboration at different (asynchronous) time.	If communication not in real-time is required.	Optional	
2.1.1	Forum	A service that provides online collaboration through discussion group, where users can exchange messages.	If service that allows exchanging messages on-line is required.	Alternative	
2.1.2	Newsgroup	A service that provides collaboration through bulletin boards, where users can put and read messages on-line or download and save messages for off-line reading.	If service that allows reading messages off-line is required.	Alternative	
…	…	…	…	…	…
3	**Workflow management feature**	**A service that includes assignment of tasks, tasks management and processing.**	**If automated management of tasks is required.**	**Optional**	
…	…	…	…	…	…

Fig. 5. Fragment of portal feature table.

3.4. *Refinement of the Feature Model*

To be used as an evaluation test example, the feature model should be represented in the form of requirements specification. First of all requirements should be derived from the feature model. We propose to use feature refinement technique for this aim. The feature refinement is a step-by-step process that involves taking the terminal features from a feature model and expanding it into software requirements, which may be expanded further into more detail requirements. In other words, it is an iterative process in which more details are considered with each pass through the requirements specification. During this process, the requirements specification consists of two parts: existing text (everything written up to this point) and intended text (everything that is to be written in the following iterations). The structure of the existing part depends on the chosen standard. We suppose in this paper that requirements are structured on the basis of ISO 9126 standard (ISO/IEC 9126, 1991). It means that refined requirements may be added to several structural parts of the specification at once.

The interface between the two parts is called backlog interface. It consists of all terms, which have already been used in the existing part, but which have not been defined yet. There are two kinds of definitions of a term: structural and functional. For example, if the term Web site has been mentioned in already written requirements, this term should be defined in a structural way, describing the required structure of this page, and in a functional way, describing the operations, which should be provided by the system for processing Web pages and for manipulation with them. Some terms require only structural definitions, some terms only functional definitions, and some should be defined in a structural as well as a functional way. All definitions should be in strong accordance with the sampling vocabulary, all should be expressed in the form of requirements, and all may include some constraints (non-functional requirements). In terms of Jackson (Jackson, 1995; Jackson, 2001), structural definitions, constraints and even some functional definitions are not real requirements, because they describe interfaces or domain properties, but not the effects that the system should produce. Similar as in the object-oriented decomposition (Rajlich and. Silva, 1988), the terms in backlog interface address objects, functions, flows and constraining properties (e.g., "interface should be *convenient* for the user"). Thus, the backlog interface contains all functions, objects, events, roles, flows and properties mentioned in the existing part of the requirements specification.

A refinement iteration step consists of the following activities:

- Select a cluster of related terms from the current backlog interface. Definition of these terms will constitute a group of functional requirements and, possibly, related groups of reliability, performance, usability or other non-functional requirements, which will be added to the existing part of the requirements specification.
- Define all terms in the cluster and add derived in such a way requirements to the corresponding part of the existing requirements specification.
- Update the backlog interface, i.e., delete all terms defined in the step, and add all new terms, which have appeared in the step.

The refinement process proceeds until all terms are defined completely without considering any design decisions. Because the requirements specification describes a generic

system, apart mandatory requirements, it should include requirements for all optional and alternative features provided by the feature model. The requirements also may be prioritised. In this paper we suppose that the MoSCoW list (Clegg *et al.*, 1994) is used to annotate requirements priorities.

The requirements are presented in a tabular form (Fig. 6). The reason to use tables is the necessity to store the example in the library of representative examples and to use tool support to manipulate with examples. For each requirement the table contains a unique identifier that reflects all previous levels of the requirements hierarchy, kind of requirement (F for functional requirements, R for reliability requirements, U for usability requirements, E for efficiency requirements, M for maintainability requirements, and P for portability requirements), statement of the requirement, requirement type (M for mandatory, O for optional, A for alternative), and requirement priority (M for must have this, S for should have this, C for could have this, and W for won't have now, but would like to have in the future). Each requirement statement should be as precise and unambiguous as possible, in an ideal case expressible in the first-order language of predicate calculus.

Fig. 7 describes example of portal requirements representation in tabular form. Because requirement statements describe a generic system, all concrete values should be replaced by parameters. For some technical reasons we also use acronyms of entity names. In the Fig. 7, parameters are written in bold and acronyms in italic.

3.5. *Evaluation Procedure*

A *quality evaluation test* consists of a test identifier, test example, test execution conditions, and expected results, which describe what the test allows to check. It is recommended that identifier include information about internal quality characteristic (e.g., functionality) under testing, the category of systems, which has been used to produce the test example (e.g., IAS for information answer systems), and about the specification language (e.g., UML) under testing. Test example is a representative set of functional and non-functional requirements that should be specified using specification language **L** under testing. Test execution conditions define what specification language is tested (e.g., Z, UML, Alloy, etc.), what tools should be used to produce specification, in which way requirements should be represented using the language **L** (each requirement should be modelled separately, groups of related requirements should be modelled, entire requirements specification should be represented as a coherent model, etc.), how requirements should be modelled (only directly, usage of language flexibility mechanisms, except the

ID	**Kind**	**Requirement**	**Type**	**Priority**
REQ - <hierarchical requirement number>	[F| R| U| E | M| P]	<Statement of the requirement>	[M| O| A]	[M| S| C| W]
…				

Fig. 6. Requirements table.

ID	Kind	Requirement	Type	Priority
REQ-1.	F	Common Web portal requirements	M	M
REQ-1.1	F	Web system must be created as Web portal **Pt**.	M	M
REQ-1.2	F	Web portal **Pt** should have three main groups of users ***Usr***: Visitor ***Vst***, Member ***Mem*** and Administrator ***Adm***.	M	S
…	…	…	…	…
REQ-2.	F	Content management	M	M
REQ-2.1	F	Portal **Pt** content items could be library items ***LbItm*** and/or document items ***DocItm***.	O	C
REQ-2.2	F	Library items ***LbItm*** should be reusable, unstructured pieces of content, typically images, static text, and banners.	O	S
…	…	…	…	…
REQ-3.	F	Collaboration requirements	O	C
REQ-3.1	F	Asynchronous collaboration requirements	O	C
REQ-3.1.1	F	Member ***Mem*** of the portal **Pt** could have possibility to participate in forums $\mathbf{Fr_1,\ldots, Fr_N}$, which are provided by the portal **Pt**.	A	C
REQ-3.1.2	F	Member ***Mem*** of the portal **Pt** could have possibility to participate in newsgroups $\mathbf{Nw_1,\ldots, Nw_N}$, which are provided by the portal **Pt**.	A	C
…	…	…	…	…

Fig. 7. Portal requirements table.

extensibility mechanisms, is allowed, etc.). Expected results describe elementary characteristics of internal quality under testing. The list of characteristics is derived from requirements specification using the following rules:

- if some requirement addresses an ontological primitive (e.g., concept) defined by sampling vocabulary (e.g., *user account*), then the characteristics "ontological sufficiency" and "ontological adequacy" should be added to the list;
- if some requirement addresses an epistemological primitive (e.g., specialisation of concept) defined by sampling questionnaire (e.g., "For every *registered user* an account should be provided."), then the characteristics "epistemological sufficiency" and "epistemological adequacy" should be added to the list;
- the characteristic "expressibility" should be always added to the list, because by definitionany requirement is a statement about the system under the consideration (e.g., "For *every* registered user an account *should be* provided.");
- if some requirements are described not explicitly and should be derived from other requirements (e.g., "Every registered user should *inherit* the access rights *from default* user and the ability to change *inherited rights* should be provided."), then the characteristic "reasoning power" should be added to the list;
- if some requirement describes composition of different ontological categories (e.g., object state and task), defined by sampling vocabulary (e.g., "*Registered user* should be able to *unsubscribe* to newsletter he has *subscribed* to before."), then the characteristic "composability" should be added to the list;

- if some requirement includes some qualified expression (e.g., "The ability to find *all news on the specified subject* that were published *during specified time interval* should be provided."), then the characteristic "selective power" should be added to the list;
- if requirements describe system at different levels of granularity (i.e., requirement specification has hierarchical structure and requirements are refined step-by-step), then the characteristic "generalitive power" should be added to the list;
- if some requirement addresses a domain-specific ontological primitive defined by sampling vocabulary that cannot be mapped directly to the type system of the specification language under testing (e.g., "All users should have unique *e-mail addresses*."), then the characteristics "extensibility" and "adaptability" should be added to the list;
- if some requirement addresses a domain-independent ontological primitive defined by sampling vocabulary defined by sampling questionnaire (e.g., *class*), then the characteristic "universality" should be added to the list.

Fig. 8 presents an example of the expected results of the test T-F-IAS-UML-01, which is developed to evaluate the ability of UML to describe information answer systems. Such tests may be used to evaluate the internal quality of a specification language for the current population of a particular category of Software Systems. To evaluate the internal quality for the whole current population of Software Systems, an appropriate test suite should be designed.

A test suite is a collection of tests developed to test some top-level group of the characteristics of internal quality (functionality, reliability, efficiency, and usability) for the

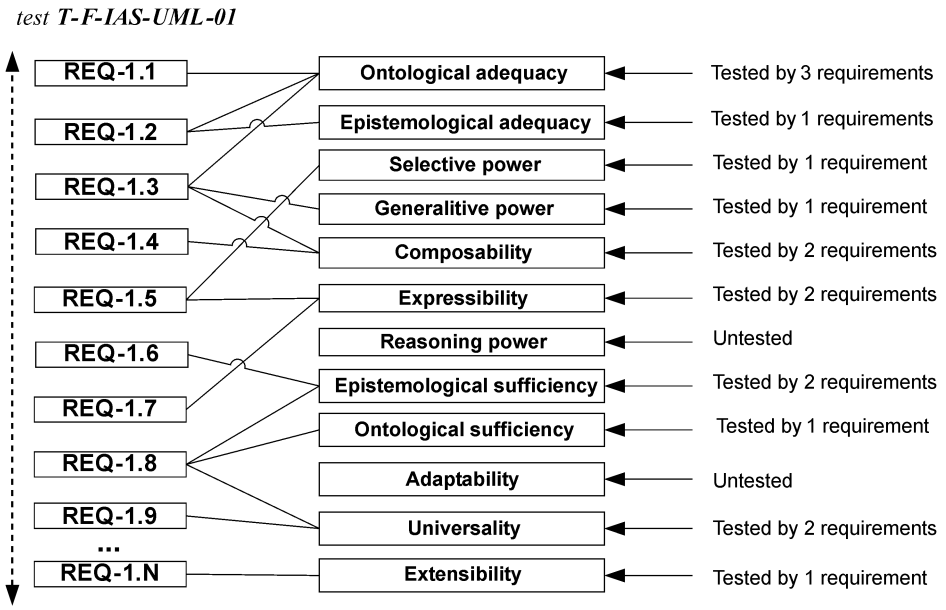

Fig. 8. An example of expected results of the test.

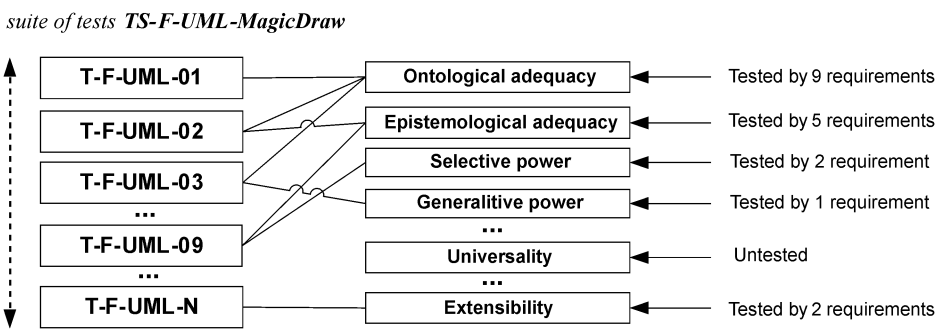

Fig. 9. An example of the coverage of elementary characteristics by the suite of evaluation tests.

particular specification language using the same testing infrastructure. It consists of evaluation tests from which each is developed to test the different category of systems. In order to minimise prior arrangement efforts, the suite should be developed for particular collection of tools required supporting the testing (i.e., for a particular testing infrastructure).

It is recommended that test suite identifier include information about internal quality characteristic (e.g., functionality) under testing, about the specification language (e.g., UML) under testing, and about for testing used tool (e.g., MagicDraw UML).

Fig. 9 presents an example of the coverage of elementary characteristics of the functionality of a specification language by the suite of evaluation tests TS-F-UML-MagicDraw, which is designed to test the functionality of UML using the CASE tool MagicDraw™[1]. In this example the numbers of requirements used to test the characteristics of functionality is calculated summarising requirements used for this aim in the tests of the suite. If the suite does not cover some characteristics, the sampling frames for each category of systems should be carefully examined. If some frames have been defined incorrectly, they should be extended to include additional systems. In the case, when all frames have been defined correctly, the uncovered characteristics should be eliminated from the further evaluation process, because they are insignificant for the current population of Software Systems. Although, at least in the ideal case, tools used to produce specifications to test some specification language cannot affect the evaluation results, it is reasonable to point out the tool for the suite explicitly, because appropriate testing infrastructure should be prepared. The test suites developed for a particular tool should cover all four top-level characteristics of the evaluated language and all should be included in the evaluation plan of this language. To avoid measurement errors, which can occur as a result of using a particular tool, it is recommended that the evaluation plan provide at least three groups of suites developed for different tools.

Apart the set of the evaluation test suites, the evaluation plan should provide time and financial constraints, evaluators and other necessary information. The coverage of tests by quality evaluation test suites for characteristics of internal quality should be described

[1]MagicDraw is the trademark of No Magic, Inc.

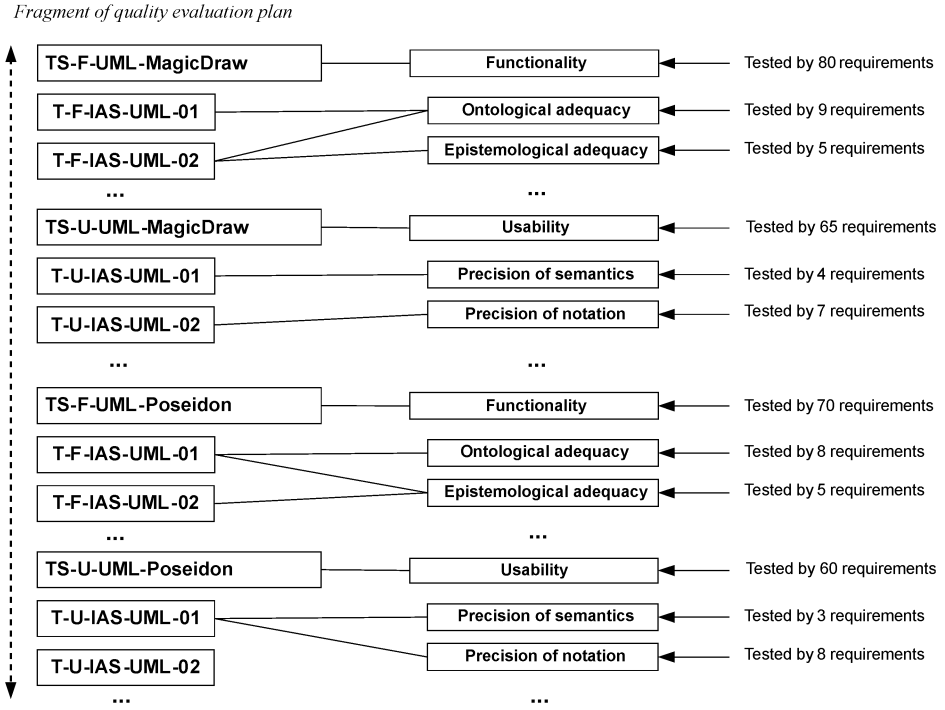

Fig. 10. An example of the coverage of tests by quality evaluation test suites.

by diagram (Fig. 10) that is produced summarising expected results of all evaluation test suites provided by the plan.

3.6. *Interpretation of Evaluation Results*

Each elementary characteristic of functionality characterises corresponding feature of a language. In our approach, the values of these characteristics describe probabilities that the corresponding features allow to specify any theoretically possible system that belongs to the current population of Software Systems. Thus, in order to define these probabilities, the specification language evaluation results should be interpreted in an appropriate way. Let us denote the feature of the language $\mathbf{L}$ described by characteristic ξ by $L(\xi)$, the probability that $L(\xi)$ will become necessary specifying the current population of Software Systems by $q(\xi)$, and the probability that $L(\xi)$ will be sufficient for this aim by $p(\xi)$. Then the value of the characteristic ξ is calculated as a production of probabilities $q(\xi)$ and $p(\xi)$.

For the characteristics of reliability, efficiency and usability $q(\xi) = 1$. For the characteristics of functionality the value of $q(\xi)$ should be defined in two steps: firstly, the value of $q(\xi)$ should be defined for each test and after that the value of $q(\xi)$ should be calculated for the whole suite taking into account all evaluation tests included in this suite.

The value of $q(\xi)$ for a particular test should be defined taking into account test coverage and the types of requirements used to test the characteristic ξ. Any characteristic ξ is tested using a group G_ξ of $N(\xi)$ requirements. If for some evaluation test G_ξ is empty, then $q(\xi) = 0$, because it means that ξ was not required to specify any requirement. In other cases the type of requirement used to test ξ should be examined:

- If at least one mandatory requirement or at least one group of alternative requirements belongs to the group G_ξ, then $q(\xi) = 1$.
- If the group G_ξ consists only of optional requirements, then, taking into account the whole feature model, probability $q_i(\xi)$ for each requirement r_i belonging to the G_ξ should be calculated, and after this $q(\xi)$ is defined as follows:

$$q(\xi) = 1 - \prod_{i=1}^{n_\xi} \big(1 - q_i(\xi)\big). \tag{1}$$

In order to define probability $q_i(\xi)$ for the r_i, it is necessary to start from the initial node of the feature model and proceed down the feature model up to the terminal feature that generates r_i. If this path is unique, then in each optional node the probability of this node should be calculated as the ratio of the number of systems, which provide corresponding optional feature, to the total number of sampled systems, and the probability for the r_i should be calculated multiplying all intermediate probabilities. If some intermediate node can be reached via several paths, then the probabilities should be calculated for each path (including the node itself), and the probability for this node should be calculated in the same way as in formula (1). After this, it is necessary to take this node as the initial node and to proceed down further. So, in this way the probabilities for all terminal features are calculated. Requirements generated by these features inherit their probabilities.

To calculate the value of $q(\xi)$ for the whole suite taking into account all evaluation tests included in this suite the formula analogous to the formula (1) should be used once again.

The probability $p(\xi)$ is defined examining testing results. It is defined as a ratio of the number $N_+(\xi)$ of requirements, which belong to the group G_ξ and has been successfully expressed in the language **L**, to the total number of requirements $N(\xi)$ in this group, i.e.:

$$p(\xi) = N_+(\xi)/N(\xi). \tag{2}$$

3.7. *Experimental Results*

The proposed methodology has been approved evaluating functional characteristics of internal quality for Z (Spivey, 1992; Woodcock, 1996) and UML 2.0 (OMG, 2005) languages. The Web portal has been chosen as a representative example for this aim, because it is a multi-frame system that can be described using two different information answer frames (for content management and for search) and two different connection frames (for chatting and workflow management). As a testing infrastructure MagicDraw UML

11.5 (No Magic, 2006) and Z/EVES 2.1 (Saaltink, 1999) have been used. The experiment demonstrates that, although the ontological and epistemological adequacies of the Z language are very low, it has high enough semantic sufficiency. The flexibility of the Z language is very low, but it is partly compensated by its relatively high completeness. On the other hand, although UML 2.0 has high enough ontological and epistemological adequacies its expressive adequacy is not very high, because of the relatively low selective power. The composability of UML 2.0 and its reasoning power also are lower than of the Z language. UML 2.0 has higher extensibility than of the Z language. On the other hand, Z language is more expressible and allows expressing more classes of formulas than UML 2.0.

3.8. *Conclusions*

The main conclusion of the paper is that there exists no simple way to evaluate functionality characteristics of internal quality of specification languages. It is a hard and complicated task, which requires relatively high time and labour overheads. Many and long-time efforts are needed to do sampling, develop test suites, test language and interpret obtained results. Besides, some difficulties of theoretical character should be overcome. The theory of problem frames is relatively new and still not sufficiently elaborated. There are no any well-grounded methods for framing and sampling particular category of systems. Too little is known how to eliminate the impact of human factor to results of evaluation procedure. On the other hand, the carried out experiment demonstrates that even evaluation of particular aspects of a particular specification language allows preparing of valuable recommendations how to use the language in more appropriate way as well as how to improve it. In addition, the systematic evaluation of internal quality not only allows identification of shortcomings and strengths of the current specification languages and evaluation of their quality in use, but also clarifies deep internal structure of each evaluated language as well as of specification languages in general and provides valuable experience that could be used during construction of new specification languages. We believe that a long-time research program for evaluation of specification languages quality is purposeful and that such program will significantly contribute to the entire theory of specification, and even programming languages.

References

Bray, I.K. (2002). *An Introduction to Requirements Engineering*. Addison-Wesley.

Bray, I.K., and K. Cox (2003). The Simulator; another, elementary problem frame? In *Pre-Proceedings of the 9th International Workshop on Requirements Engineering – Foundation For Software Quality*, *In Conjunction with CAiSE'03*. Klagenfurt/Velden, Austria. pp. 101–104. Available at: `http://crinfo.univ-paris1.fr/REFSQ/03/papers/REFSQ03-PreProceedings.pdf`

Caplinskas, A., and J. Gasperovic (2005a). An approach to evaluate quality in use of IS specification language. In J. Barzdins and A. Caplinskas (Eds.), *Frontiers in Artificial Intelligence and Applications*, vol. 118. IOS Press, Amsterdam. pp. 152–166.

Caplinskas, A., and J. Gasperovic (2005b). Functionality of information systems specification language: concept, evaluation methodology, and evaluation problems. In O. Vasilecas *et al.* (Eds.), *Information Systems Development. Advances in Theory, Practice and Education*. Kluwer Plenum Publishers. pp. 341–351.

Caplinskas, A., and J. Gasperovic (2005c). Technique to aggregate the characteristics of internal quality of an IS specification language. *Informatica*, **16**(4), 519–540. Available at: `http://www.vtex.lt/Informatica/Contents.htm`

Clegg, D., and B. Richard (1994). *Case Method Fast-Track. A RAD Approach*. Addison Wesley.

Cochran, G. (1977). *Sampling Techniques*. 3rd ed. John Wiley and Sons.

ISO/IEC 14598-1:1999 (1999). *Information Technology – Software Product Evaluation – Part 1: General Overview*. First edition, 1999-04-15.

ISO/IEC 9126 (1991). *Information Technology – Software Product Evaluation – Quality Characteristics and Guidelines for their Use*. First edition, 1991-12-15, reference number ISO/IEC 9126: 1991(E).

Jackson, D. (1999). *Comparison of Object Modelling Notations: Alloy, UML and Z*. MIT Lab for Computer Science. Available at: `http://geyer.lcs.mit.edu/~dnj/pubs/alloy~comparison.pdf`

Jackson, M. (1995). *Software Requirements and Specifications*. Addison-Wesley.

Jackson, M. (2001). *Problem Frames*. Addison-Wesley.

Jackson, M. (2000). Problem analysis and structure. In *Proceedings of NATO Summer School*, Marktoberdorf.

Kang, K., S. Cohen, J. Hess, W. Novak and S. Peterson (1990). *Feature-Oriented Domain Analysis* (*FODA*) *Feasibility Study*. Technical Report. CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh.

Krogstie, J. (2003). Evaluating UML using a generic quality framework. In: L. Favre (Ed.), *UML and the Unified Process*. Idea Group Publishing, Hershey, PA, USA. pp. 1–22.

Lindland, O.I., G. Sindre and A. Sølvberg (1994). Understanding quality in conceptual modelling. *IEEE Software*, **11**(2), 42–49.

Milton, S., E. Kazmierczak and C. Keen (1998). Comparing data modelling frameworks using Chisholm's ontology. In J.A. Bartoli (Ed.), *Proceedings of the 4th European Conference on Information Systems*, *ECIS'98*. Aix-en-Provence, France, Euro-Arab Management School. pp. 260–272.

Mylopoulos, J. (1998). Characterizing information modeling techniques. In P. Bernus, K. Mertins, G. Schmidt (Eds.), *Handbook on Architectures of Information Systems*. Springer, Berlin. pp. 17–57.

No Magic, Inc. (2006). *MagicDraw™UML 11.5 User manual*. Available at: `http://www.magicdraw.com/main.php?ts=navig&NMSESSID=500a881a8816599a88224a0248272c6a&cmd_show=1&menu=download_manual&NMSESSID=500a881a8816599a88224a0248272c6a`

O'Brien, J.A. (2000). *Introduction to Information Systems*: *Essentials for the Internetworked Enterprise*, 9th ed. Irwin/McGraw-Hill, New York.

OMG (2005). *Unified Modeling Language: Superstructure Version 2.0*. Document formal/05-07-04, Object Management Group. Avalaible at: `www.omg.org/docs/formal/05-07-04.pdf`

Opdahl, AL. (1997). Applying semantic quality criteria to multi-perspective problem analysis methods. In E. Dubois, AL. Opdahl AL, K. Pohl (Eds.), *Proceedings of the Third International Workshop on Requirements Engineering*: *Foundations of Software Quality – REFSQ'97*. Barcelona, Catalonia, Spain. pp. 49–66.

Patton, M.Q. (1990). *Qualitative Evaluation and Research Methods*, 2nd ed. Sage Publications, Newbury Park, CA.

Rajlich, V., and J. Silva (1988). Two object oriented decomposition methods. In *Proceedings of the 5th Washington Ada Symposium on Ada*. Tyson's Corner, Virginia, USA. pp. 171–176.

Saaltink, M. (1999). *The Z/EVES 2.0 User's Guide*. TR-99-5493-06a. ORA Canada. Available at: `http://nexp.cs.pdx.edu/bart/omse/omse522-winter2002/nfp/sw/z-eves/99-5493-06a-users.pdf`

Spivey, J.M. (1992). *The Z Specification Language*, 2nd. ed. Prentice-Hall.

Trochim, W.M.K. (2002). *Research Methods Knowledge Base*. Cornell University. Available at: `http://www.socialresearchmethods.net/kb/`

Wand, Y., and R. Weber (1989). An ontological evaluation of systems analysis and design methods. In ED. Falkenberg and P. Lindgreen (Eds.), *Information Systems Concepts*: *An In-Depth Analysis*. IOS Press, North-Holland, Amsterdam. pp. 79–107.

Wand, Y., and R. Weber (1995). On the deep structure of information systems. *Information Systems Journal*, **5**, 203–223.

Woodcock, J., and J. Davies (1996). *Using Z*: *Specification, Refinement, and Proof*. Prentice-Hall.

J. Gasperovič is a doctoral student at the Institute of Mathematics and Informatics. Her research area encompasses formal, semi-formal and lightweight formal specification languages and their application in software engineering and information systems engineering.

A. Čaplinskas is a principal researcher at the Institute of Mathematics and Informatics. The area of his scientific interest includes software engineering, information system engineering, legislative engineering, and knowledge-based systems.

Specifikavimo kalbų funkcionalumo vertinimo metodika

Jelena GASPEROVIČ, Albertas ČAPLINSKAS

Straipsnyje nagrinėjamas specifikavimo kalbų funkcionalumo vertinimo uždavinys. Apžvelgtos pagrindinės specifikavimo kalbų kokybės vertinimo problemos, pabrėžiant specifikavimo kalbų elementariųjų kokybės charakteristikų vertinimo problemą. Šiai problemai spręsti pasiūlyta ir detaliai aprašyta specifikavimo kalbų funkcionalumo charakteristikų vertinimo metodika. Trumpai aptarti rezultatai, gauti panaudojus pasiūlytąją metodiką Z ir UML kalbų funkcionalumui vertinti.