Accordance with Professional Requirements and Didactic Units in the IT Sphere*

Igor Kaftannikov, Vlada Zhernova, and Maria Lykova

South Ural State University, Chelyabinsk, Russia
kaftannikovil@susu.ru

Abstract. Currently, the requirements for professional knowledge and programmer skills in the IT sphere are determined by a variety of factors. There is a sufficiently significant differentiation of the companies requirements of different directions and majors for projects of the same subject and level. Consideration of many requirements by educational institutions of various types (academic, applied and additional) is almost impossible. There are international and Russian documents that define the levelled structuring of requirements heuristically. At the same time, the cumulative sets of components of professional knowledge and basis of requirements skills are not defined clearly. The above, as well as the lack of level structuring of the components, makes it difficult to compare the sets of knowledge components, skills requirements and the sets of didactic units of knowledge and skills provided in the disciplines and courses of educational establishments. The paper discusses the grounds and model for accordance between the components of professional requirements with educational didactic units, and also examines the possible procedure for implementing such an agreement on the example of the Russian professional standard "Programmer" and European-e-Competence-Framework-3.0 CEN CWA 16234-1 2014.

Keywords: Russian professional standard "Programmer" · European-e-Competence-Framework-3.0 · educational standards · competences · education in the IT-field

1 Basis: Sets of Knowledge and Skill Components in Russian IT Standard and E-ICT Framework

Consider the competence B.1. Application Development (European e-Competency Framework (e-CF) version 3.0 [1] and the Russian professional standard "Programmer" [2] as examples and sources of the requirements under consideration for knowledge and skills components. The competence B.1.includes 14 knowledge components and 8 skill components (Fig. 1), which are not structured by levels, although the competence presupposes the presence of 3 levels,

* The work was supported by Act 211 Government of the Russian Federation, contract 02.A03.21.0011.

but, as indicated above, the structuring of the components by levels is not provided here. "Programmer" is the professional standard of the Russian Federation and lists 15 labor functions and 67 components of knowledge for them, as well as 52 components of very general skills (programming languages, formalization and algorithmization of tasks, databases, information security, etc.), as well without any structuring of the components themselves.

Dimension 1 e-Comp. area	**B. BUILD**				
Dimension 2 e-Competence: Title + generic description	**B.1. Application Development** Interprets the application design to develop a suitable application in accordance with customer needs. Adapts existing solutions by e.g. porting an application to another operating system. Codes, debugs, tests and documents and communicates product development stages. Selects appropriate technical options for development such as reusing, improving or reconfiguration of existing components. Optimises efficiency, cost and quality. Validates results with user representatives, integrates and commissions the overall solution.				
Dimension 3 e-Competence proficiency levels e-1 to e-5, related to EQF levels 3 to 8	**Level 1**	**Level 2**	**Level 3**	**Level 4**	**Level 5**
	Acts under guidance to develop, test and document applications.	Systematically develops and validates applications.	Acts creatively to develop applications and to select appropriate technical options. Accounts for others development activities. Optimizes application development, maintenance and performance by employing design patterns and by reusing proved solutions.	–	–
Dimension 4 Knowledge examples *Knows/aware of/ familiar with*	K1 appropriate software programs/modules K2 hardware components, tools and hardware architectures K3 functional & technical designing K4 state of the art technologies K5 programming languages K6 Power consumption models of software and/or hardware K7 DBMS K8 operating Systems and software platforms K9 Integrated development environment (IDE) K10 rapid application development (RAD) K11 IPR issues K12 modeling technology and languages K13 interface definition languages (IDL) K14 security				
Skills examples *Is able to*	S1 explain and communicate the design/development to the customer S2 perform and evaluate test results against product specifications S3 apply appropriate software and/or hardware architectures S4 develop user interfaces, business software components and embedded software components S5 manage and guarantee high levels of cohesion and quality S6 use data models S7 perform and evaluate test in the customer or target environment S8 cooperate with development team and with application designers				

Fig. 1. Structure and components of competence B.1. Application Development

It makes sense to consider the cumulative set of requirements for both documents, with the capacity of 81 knowledge components and 60 skill components given the trends and requirements of global staff and academic mobility. Also, there is the change in the power of sets with the addition of requirements for other documents or their localization. So the procedures for forming groups of

sets and mapping their requirements to educational programs must be sufficiently universal. Besides as the relations of order are not defined on the sets under consideration (this does not make sense), the level structuring of competencies and requirements of the standard should be provided by the level structure of their components. There is the correspondence in the level structure of educational programs presented which are in the sequence of didactic units.

Define the objects under consideration, their grouping, properties and relations for them to formalize the above procedures. We assume that EAC is the set of knowledge components from the "European e-Competence Framework (e-CF) version 3.0". Set: $EAC = \{eAC_i^j\}$ with AC is an aggregated component, e is localization prefix (document reference), j is component index in competence or standard, i is index of competence or standard.

We assume RAC is set of aggregated components of knowledge from the Russian standard "Programmer". Set: $RAC = \{rAC_j^i\}$.

Similarly, we assume the set of aggregated skills components EAS and RAS. Set: $EAS = \{eAS_j^i\}$, $RAS = \{rAS_j^i\}$

We assume IAC is sets of components from all considered documents will be called integrated. Set: $IAC = EAC \cup RAC = \{eAC_j^i, rAC_j^i\}$ or $IAC = \{AC_m^n\} = \{eAC_j^i\} \cup \{rAC_j^i\}$; $IAS = EAS \cup RAS = \{AS_m^n\} = \{eAS_j^i, rAS_j^i\} = \{eAS_j^i\} \cup \{rAS_j^i\}$.

Individual analysis of each of AC_m^n, AS_m^n is not constructive, due to the fact that the subject areas that are indistinctly defined by the names of components have many intersections due to their inclusion in the more global subject area "Application Development, Programmer". (e.g.: *programming languages, languages, utilities and programming environments, software development methodologies,* etc.). Therefore, in our opinion, it is preferable to pre-decompose IAC, IAS into subsets including aggregated components closest to the semantics of functional purpose (subject domains), allowing to determine common metrics for all AC_m^n, AS_m^n, included in them. We assume such sets as grouping: GAC, GAS. Wherein: $GACt \in IAC$, $GASt \in IAS$.

For example, GAC_1 is a set including AC_5, AC_6:

AC_5	Typical software metrics
AC_6	The main methods for measuring and evaluating the characteristics of software

and defines the subject area "Characteristics and metrics of software".

GAC_3 includes:

AC_6	Methods and methods of formalizing tasks
AC_7	Methods and techniques for algorithmizing tasks
AC_8	Algorithms for solving typical problems, areas and methods of their application

And defines the subject area "Methods and techniques of algorithmizing and formalizing tasks".

Level structuring of directly grouping components (GAC, GAS) can be performed only through the complex expert formation of the relations of order due

to the fuzzy definition of the domain. At the same time a simpler method of implementing the level structuring of each group can be the decomposition of the group as a whole into components (call them CC, CS). Each of such defines is a much more specific domain. For example, programming languages, software metrics for which metrics can be formed according to certain procedures, which can be either formal or pseudo formal, as required.

2 METHODS AND PROCEDURES OF LEVEL STRUCTURIZATION

Proposed methods and procedures are considered in the example of the CC "Programming Languages". A component of this type is called composite, firstly, due to the fact that set of the programming languages numbers several hundred, (maximum - more than 700). At the same time, several new ones appear each year. Naturally, there is also a degradation in the use (dying out) of languages and the current group of leaders, which can be determined using ratings generated by certain resources that are significant for the IT community and, at the same time, using these ratings to form the metrics applicable to building the level structure of components. For the programming languages component, there are several most significant resources. In this case we used the following three:

1. GitHub: https://github.com/, ranking by the number of lines of code;
2. TIOBE Programming Community Index: https://tiobe.com, the ranking on the popularity of the language, expressed in the number of search queries;
3. The RedMonk Programming Language Rankings: http://redmonk.com, a comprehensive rating.

According to these sources (at the beginning of 2017) the following rating table was compiled (Table 1) for the 15 most popular programming languages for each resource:

In all three ratings, the following fifteen programming languages are present: C, C++, C#, Java, JavaScript, PHP, Python, Ruby, Go. These languages are used today in virtually all software technologies and platforms, while programming most components of software systems.

In our opinion this means that the verification of knowledge of programming languages can be done on the basis of knowledge of the languages listed above, since other (close) languages can be quite quickly mastered, with a good knowledge of the above. Naturally, it is possible to specialize in languages as well as dynamic changes in ratings and entries in the tops of popularity. This means that the tiered system of knowledge components must also be easily modernized, migratory and expandable. Secondly, the components of knowledge consist not only of the current set of the most popular languages, but must take into account their structure, properties, application features, and so on. in our opinion the second component of knowledge components is quite fully reflected in good

Table 1. Popular program languages

Github	TIOBE	Redmonk
JavaScript	Java	JavaScript
Java	C	Java
Python	C++	PHP
Ruby	C#	Python
PHP	Python	C#
C++	Visual Basic .NET	C++
CSS	Javascript	Ruby
C#	Perl	CSS
C	ASM	C
Go	PHP	Objective-C
Shell	Delphi/Object Pascal	Shell
Objective-C	Ruby	R
Scala	Go	Perl
Swift	Swift	Scala
Typescript	Visual Basic	Go

(most popular) textbooks and monographs on languages. Similarly, the selection of the textbooks themselves is simple enough to provide for popularity, taking into account the opinion of experts.

Assuming the use of 5 levels of structuring according to [1], we, used junior / middle level textbooks or reference type books. In our opinion these books usually include knowledge and skills in programming languages corresponding to the first three levels of structuring, as they are designed to master the basic capabilities of the language or present a review of it.

Further, the reconciliation procedure included the selection of books by analyzing the subjective estimates of users obtained using professional resources and forums (**habrahabr**, **tproger**, **geektimes**, **toster**, **github**, **stackoverflow**, etc.). In order to simplify the analysis and publication so far we used only one language (C ++) and four most popular Russian-language and translated publications:

1. Podbelsky V.V. C ++ language;
2. Straustrup B. The programming language C ++;
3. Prata S. C ++ Programming Language;
4. Eckel B. Philosophy of C ++.

Obviously, similar procedures can be performed with respect to other programming languages. Naturally, the number and list of publications can be changed. For each book an enlarged table of contents (chapters) was taken for consideration, which can be defined as the didactic units of the corresponding work program. The follow table of contents is a kind of information about a language structured according to the complexity of mastering ("from simple to complex").

Expert structuring on 1-3 levels (and also, in some cases, for 4 levels) is presented in Figure 2 for each source separately.

Podbelsky V.V. C ++ language	Level
1. Overview \ introduction to the language	1
2. Lexical Basics	1
3. Scalar types and expressions	1
4. Language operators	1
5. Addresses, pointers, arrays, memory	2
6. Functions, pointers, links	1.2
7. Structures and associations	2
8. Preprocessor tools	2
9. Class as an abstract type	2
10. Inheritance and other opportunities	2.3
11. Input / Output	2.3
12. Exception handling	3

S. Prata Programming language C ++	Level
1-2 Introduction	1
3. Work with data (types)	1
4. Compound types (arrays, strings, structures, pointers)	2
5. Cycles and expressions of relations	1
6. Branching operators and logical operations	1
7-8 Functions	1.2
9. Models of memory and namespaces	2
10-11 Objects and Classes	2
12. Classes and dynamic memory allocation	3
13. Inheritance	2.3
14. Code reuse	3
15. Friendliness, exceptions, etc.	3
16. String class and standard template library (STL)	3.4
17. Input / Output	2.3

B. Stroustrup C ++ programming language	Level
1-3 Overview of the language and standard library	1
4. Types and ads	1
5. Pointers, arrays, structures	2
6. Expressions and instructions	1
7. Functions	1.2
8. Namespaces and Exceptions	2
9. Source files of the program	1
10. Classes	2
11. Operator overloading	2

B. Eckel Philosophy of C ++	Level
1. Introduction to objects	1
2. Creating and using objects	1
3. Elements of C in C ++	1.2
4. Abstract representation of data	2
5. Hiding the implementation	2
6. Initialization and cleaning	2
7. Function Overloading and Default Arguments	3

Fig. 2. Structure of the tutorial

The structuring of levels 4 and 5 seems more complicated because it involves expanding the objects of the subject areas of languages, comparing their relationships and relationships, as well as existing practical experience. The above books do not include didactic units related to these levels. Therefore, to structure the knowledge and skills of 4-5 levels, it is necessary to consider the literature on specialized extensions / libraries / programming languages frameworks, since

the standard tools of any language are not sufficient for writing projects other than educational ones.

It should also be taken into account that the structuring of knowledge of levels 4 and 5 will be specific for each language separately. If for the first three levels it is possible, in some approximation, to allocate opportunities and features inherent in the whole group of languages, and thus abstract from the choice of language, then for the 4th and 5th level, it is necessary to choose a specific technology (a set of libraries, a framework) evaluation.

The choice of technology, in our opinion, should be based on the specifics of the project or subject area (web, mobile, desktop, etc.). This will allow more fully assess to the knowledge of a specialist for a specific application task.

Also one should note one more difference between 4 and 5 level. If level 4 assumes knowledge of both language and applied technology (libraries, framework), then level 5 can be defined as the level of development of the full-stack project. Full-stack is development from scratch, a fully operational project. That is, a specialist with level 5, apart from having knowledge of the language and the chosen technology, is competent and capable of creating a workable project from scratch using these technologies, and is also competent in managing the team / developer department to create a project from scratch in this area using the language and the chosen technology.

Competence and the "Programmer" standard were taken by us as an example of one of the broadest subject areas with a wide range of requirements, both in terms of knowledge and skills. In this example, we hope to demonstrate the principle possibility (using the proposed methodology and procedures) to create a system of structured level components of GAC, GAS consisting of a structured level system of knowledge and skills of the "lower" level that are part of CK, CS and are the fundamental basis For a certain formalization (pseudo formalization) of the requirements of the subject area of professional qualification (competence). Also, at the same time, the methodology of professional requirements transformation into didactic units of the training, retraining, advanced training programs and, accordingly, a certain, logically justified harmonization of professional requirements and educational programs was proposed. It should also be noted that expert opinions and approvals at all levels are supposed to be conducted with the involvement of representatives of the business community, for which the necessary procedures are determined at the computer department of the SUSU and a corresponding portal is being developed.

References

1. Europeane-CompetenceFramework (e-CF) version 3.0; http://www.ecompetences.eu/.
2. Professional Standard of the Russian Federation "Programmer"; http://www.rosmintrud.ru/docs/mintrud/orders/138/.