Competency Development in the Object-oriented Programming Style Education

Muharem Mollov, Gencho Stoitsov

University of Plovdiv "Paisii Hilendarski", 236 Bulgaria Blvd., 4003 Plovdiv, Bulgaria

Abstract **– The article presents a spiral approach for the competency development in the object-oriented programming style education as a part of the occupation "Application programmer" of the national program "Education for IT career" (NPEITC) in the Bulgarian Ministry of Education and Science. The education is realized by a system of tasks comprising topically linked modules of the study plan. Such system is demonstrated and the obtained results from its application, followed by the analyses confirm the success of the competency approach, are presented.**

Keywords **– object-oriented programming, competencies, spiral approach.**

1. Introduction

The concept of competency approach is met in the field of education since early 1960s. Robert White defines competency as basic motive for acquiring knowledge and skills [1]. McClelland use the term competency in the sense of successful combination of knowledge, skills, attitude and behavior of workers to achieve desired results in a given professional field and organization [2].

DOI: 10.18421/TEM104-59
https://doi.org/10.18421/TEM104-59

Corresponding author: Genho Stoitsov,
University of Plovdiv "Paisii Hilendarski", 236 Bulgaria Blvd., 4003 Plovdiv, Bulgaria.
Email: stoitzov@uni-plovdiv.bg

Received: 24 September 2021.
Revised: 09 November 2021.
Accepted: 15 November 2021.
Published: 26 November 2021.

© 2021 Muharem Mollov & Gencho Stoitsov; published by UIKTEN. This work is licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 4.0 License.

The article is published with Open Access at www.temjournal.com

Boyatzis [3] suggests new strategy in the field of human resources management on the base of competency concept. Raven describes competencies as "motivated capabilities" [4].

Modern high school education meets many challenges which require new approaches and strategies to be applied for its realization. It is synchronized with the European Qualification Framework (EQF), created by the European Union (EU), which relates the national qualifications, assists the transborder mobility of students and workers and stimulates the lifelong learning and the professional development in Europe. Along with the EQF, the EU creates many different competency frames. For example, some of the digital competency frames are: Digital Competence Framework for citizens (DigComp 2.1), European e-Competence Framework (e-CF) for the ICT professionals, European Framework for the Digital Competence of Educators: (DigCompEdu). They are generally accepted standards which guarantee minimal mandatory requirements for objects of standardization [5]. In this context, representatives from the Bulgarian ICT sector unite around the idea for accepting one standard for the employment of candidates in software industry and suggest a frame, based on EQF and Comité Européen de Normalisation Workshop Agreement (CWA 16458:2012). The National Agency for Vocational Education and Training (NAVET) creates such national educational standard (NES) for the occupation "Application programmer" according to these recommendations.

2. Aim of the Research

The aim of the research is a competency approach based on spiral competency development to be applied in the education modules "Introduction in object-oriented programming" (IOOP) and "Object-oriented programming" (OOP) [6]. The results from the conducted competency-based education to be analyzed and the conclusions are to be made.

3. Competency Approach in the Education for the Occupation "Application programmer"

The definition about education was changed many times in the history. Nowadays, it is synchronized with the modern concept of competency approach concerning the lifelong learning.

Based on the relations shown in Figure 1, the competency approach takes part in the education process for the occupation Application programmer. The main aim of professional education is the effective preparation of students for the labor market. Employers need specialists with a particular set of acquired knowledge, skills and competencies, described in job characteristics and employment requirements.

The learning outcomes for the occupation Application programmer are defined in NES according to:

- the bachelor and master degrees of Association for Computing Machinery (ACM) for Computer Education;
- the ICT sector requirements.

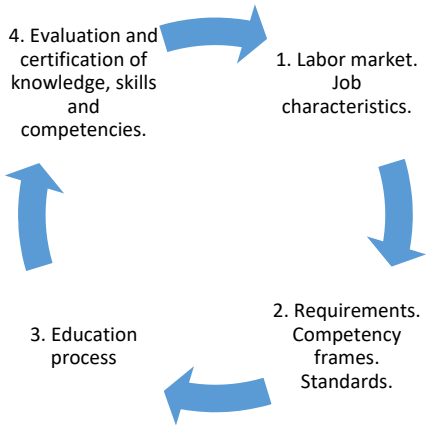

Figure 1. Competency life cycle

NES introduces a list of learning result units (LRU) for general professional preparation and specialized preparation. LRU are not requirements but rather they are standards for the acquirement of expected knowledge, skills and competencies. The education follows the study plan for the occupation Application programmer the content of which is based on the criteria described in the national standard. Some of the education modules included in the national standard are: Introduction in programming (IP), Programming (P), Introduction in OOP, OOP, Introduction in algorithms and data structures (IADS), Algorithms and data structures (ADS), Databases, Software development, Integrated systems and operation systems, Production practice, Functional programming, Internet programming, and Software engineering. Knowledge, skills and competencies are the main targets of evaluation which is conducted on levels: study hour, module exam, final exam. Business representatives are recommended to participate in education and evaluation.

4. Study

The study focuses on the students' competency development in using object-oriented programming style and basic-level functional programming style during the education modules IOOP and OOP. IOOP reviews the basics of object-oriented programming style concerning classes and objects while OOP helps students to get familiar with the OOP principles-Encapsulation, Abstraction, Inheritance, and Polymorphism. A system of tasks, demonstrating the application of spiral education approach for upgrading the specific knowledge, skills and competencies which students already developed for the occupation "Application programmer", is suggested. One of the module pairs (IP and P, IOOP and OOP, IADS and ADS) – IOOP and OOP, is chosen.

5. Instrumentation

Tasks with a gradual increase in difficulty are used. Solving the easier tasks is one step ahead for finding the solution of the next tasks. They include the knowledge, skills and competencies for OOP paradigm, set out in LRU 10 of NES. The last task aims also the competency development in using functional programming style (FPS). Solving the practical tasks contributes to the enrichment of knowledge and the acquirement of skills needed for using OOP and FPS. Additionally, the development of competency for proceeding hierarchical datasets is achieved. The applied instrumentation and approaches for finding the right solutions of the tasks are chosen by the students. The extent to which the students can cope with a given problem defines the students' competency. The competency evaluation is holistic which means it cannot allow an absolute comparison between the students to be made. Only if the evaluation conditions are equal for all students, then a relative comparison between them is allowed.

The system includes such four tasks from different topics of the both modules. The acquired knowledge and skills from the previous tasks along with the knowledge, skills and competencies needed for using the integrated data structure *List* in the library *System Collections. Generic* created with the help of programming language C# are used for solving the tasks. In addition, some basic LINQ functionalities, studied in the second module of the study plan of NPEITC are necessary to be applied for finding the solutions of the tasks. The last task contains all components of OOP - Encapsulation, Abstraction, Inheritance and Polymorphism.

During the learning of the OOP paradigm, the modelling of objects and object relationships, which are with different degree of difficulty and abstraction, is essentially important [7]. This suggests a gradual increase in the task difficulty and abstraction levels by which students study deeply the basic principles of encapsulation and especially of inheritance,

abstraction and polymorphism along with good practices of their analysis, design and implementation [8].

System of Tasks

Task 1. Person

Create class *Person* with fields - *string* ***name*** and *int* ***age***. Insert person's ***name*** and ***age*** on two rows, then the program creates an object and if the data is valid, the following sentence appears:

I am {name} and I am {age}-year old.

Task 2. Students' personal data validation

Create *class* ***Student***, inheriting the *class* ***Person*** in ***Task 1***, with fields *int* ***grade*** and *double* ***success***. Let there be the following limits: ***name*** mustn't be shorter than 3 symbols; ***age*** to be between 7 and 19. ***grade*** to be between 1 and 12; ***success*** to be between 2 and 6. After insertion of wrong data, the program generates exceptions exporting suitable messages.

Task 3. Grade

Add *class* ***Group***, which contains a student list from a grade in text form. The following format is used for data insertion:

<name> <age> <grade> <success>|...|<name> <age> <grade> <success>

The sign for separating the individual data of each student from the other students' data is „|". Meanwhile, interval is used for the separation of the data of a given student.

Task 4. School

Create a program by inserting data for students and teachers on one row. Use the sign "|" for separating the individual data of each person from the others' data and interval for the separation of the data of a given person. The following format for students' and teachers' data is used:

Student <name> <age> <grade> <success>
Teacher <name> <age> <speciality> <salary>

For example:

Student Ivan 13 6 4.5|Teacher Alex 35 Mathematics 1340.56|Student Ana 11 2 4.1|Student Iva 3 4 5.5|Teacher Alexander 30 Informatics 1366.56

this program should realize the following functionalities:

1. To export data for people according to their insertion order;
2. To export filtered data for students according to their insertion order;
3. To export filtered data for teachers according to their insertion order.

Limits: Personal names are to be at least 3 symbols. Students' age is to be between 7 and 19 years. The teachers' age is to be above 23 years. Students' grade is to be between 1 and 12. Students' success is to be between 2 and 6. Teacher salary cannot be a number lower than 700. After insertion of wrong data, the program generates exceptions exporting suitable messages.

6. Analysis and Results

During the research, it is observed that the students can find the solution of the first task with easiness. They share that this task helps them to understand the meaning of object states (property, data, object characteristics), the characteristic object actions, the ways for instantiating objects and the application of objects in coding. The students can realize the definitions of *class* and instance of *class*.

```
public class Person
   {
      private string name;
      private int age;

      public string Name
      {
         get
         {
            return this.name;
         }
         set
         {
            this.name = value;
         }
      }
      public int Age
      {
         get
         {
            return this.age;
         }
         set
         {
            this.age = value;
         }
      }
      public Person(string name, int age)
      {
         this.Name = name;
         this.Age = age;
      }
      public void Print()
      {
         Console.WriteLine($"I am {this.Name} and I am
{this.Age}-years old.");
      }
   }
```

Figure 2. Task 1 - class Person

The students acquire the knowledge and skills for data encapsulation and develop methods in accordance with the encapsulation principles. The method Print() gives a feedback for the fields name and age (Figure 2) through the properties Name and Age. It is called by a defined object from the *class* ***Person*** of the main program.

During ***Task 2***, the students can apply the acquired knowledge for exceptions, which are studied in module 2 - "Programming", to validate data. This can be accepted as competency development.

To define the *class* ***Student***, an inheritance of the *abstract class* ***Person*** is used in the task. The *class* ***Person*** from ***Task 1*** is modified in the setters of the properties ***Name*** and ***Age*** to allow the realization of the validation requirements.

```
public abstract class Person
   {
      private string name;
      private int age;

      public string Name
      {
         get
         {
            return this.name;
         }
         set
         {
            if (value.Length < 3)
            {
               throw new ArgumentException("Name cannot
be shorter than 3 charcters");
            }
            this.name = value;
         }
      }
      public virtual int Age
      {
         get
         {
            return this.age;
         }

         set
         {
           this.age = value;
         }
      }
      public Person(string name, int age)
      {
         this.Name = name;
         this.Age = age;
      }
      public Person()
      {
      }
      public override string ToString()
      {
         return $"I am {this.name} and I am {this.age}-
years old.";
      }
   }
```

Figure 3. Task 2 - abstract class Person

The already gained knowledge about virtual methods, inheritance and abstraction helps the students to virtualize the method ***ToString(),*** in which the functionality of the method ***Print()*** was realized in the previous task, to export data (*Figure 3*).

Inheritance and polymorphism are realized in the *class* ***Student***. The class constructor calls up the basic class constructor and initialize the remaining fields by using the field properties. Subclass validation is also implemented. The method ***ToString()*** interacts with the basic method by the construction ***base()*** (*Figure 4*).

```
public class Student: Person
   {
      private int grade;
      private double success;
      public int Grade
      {
         get
         {
            return this.grade;
         }
         set
         {

            if (value < 1 || value > 12)
            {
               throw new
System.ArgumentOutOfRangeException("Grade", "Grade
cannot be less than 1 or more than 12.");
            }
            this.grade = value;
         }
      }

      public override int Age
      {
         get => base.Age;
         set
         {
            if (value < 7 || value > 19)
            {
               throw new
ArgumentOutOfRangeException("Student's Age", "Age
cannot be less than 7 or greater than 19");
            }
            base.Age = value;
         }
      }
      public double Success
      {
         get
         {
            return this.success;
         }
         set
         {

            if (value < 2 || value > 6)
            {
```

```
                throw new ArgumentOutOfRangeException
("Student Success", "Success can not be less than 2 or
more than 6.");
            }
            this.success = value;
        }
    }
    public Student(string name, int age, int grade, double
success)
        : base(name, age)
    {
        this.Grade = grade;
        this.Success = success;
    }
    public override string ToString()
    {
        return $" {base.ToString()} "+
            $"I am a student in {this.grade} grade, " +
            $"my success is {this.success:F2}.";
    }
}
```

Figure 4. Task 2 - class Student

The main program creates an object from the *class* Student and generates exceptions in the constructors of the basic or inheriting class in the case of invalid data insertion.

In Task 3, the students can use the competencies developed in Module 2 to work with collections and basic features of the LINQ library and the knowledge and skills from Task 1 and Task 2 to upgrade the competency for working with object collections. As a result, almost all of the students managed to solve the task successfully by themselves.

The *class* ***Group*** (*Figure 5*) includes an object collection as a field. The override ***ToString()*** export data for all students by using the ***ToString()*** method of the *class* ***Student***.

```
class Group
  {

    private List<Student> students = new
List<Student>();

    public List<Student> Students { get => students; set
=> students = value; }

    public Group(List<Student> students)
    {
        this.Students = students;
    }
    public Group()
    {

    }

    public override string ToString()
    {
        string t ="The group members are:\n";
        foreach (var st in this.Students)
        {
            t += st.ToString();
        }
        return t;
    }
}
```

Figure 5. Task 3 - class Group

The all knowledge, skills and competencies acquired from the previous tasks are used for solving the fourth task. In this context, the aim of the task is the creation of a small information system for teachers and students in a school. The developed students' competencies in working with objects and object hierarchies, instantiating objects in hierarchies, data validation and proceeding find application in this task. Using the LINQ library helps the students to write a FPS code for the proceeding of collection elements. The following task features are implemented:

- data validation by the use of exceptions;
- creation of tree data structures by applying polymorphism for the instantiation of elements in a collection with the help of objects and classes;
- data export for each collection element;
- data export for each collection element based on given criteria for it.

A new *class* ***Teacher*** is added to the object hierarchy (*Figure 6*).

```
public class Teacher : Person
  {
    private string position;
    private double salary;

    public string Position
    {
        get
        {
            return this.position;
        }
        set
        {
            if (value.Length < 2)
            {
                throw new
System.ArgumentOutOfRangeException("Teacher's
Position", "Position can not be less than 2 symbols");
            }
            this.position = value;
        }
    }
    public double Salary
    {
        get
        {
            return this.salary;
        }
        set
```

```
        {
            if (value < 700)
            {
                throw new
System.ArgumentOutOfRangeException ("Teacher's
Salary", "Salary can not be less than 700");
            }
            this.salary = value;
        }
    }
    public override int Age
    {
        get => base.Age;
        set
        {
            if (value < 23)
            {
                throw new
ArgumentOutOfRangeException("Teacher's age", "Age
cannot be less than 23");
            }
            base.Age = value;
        }
    }
    public Teacher(string name, int age, string position,
double salary)
        : base(name, age)
    {
        this.Position = position;
        this.Salary = salary;
    }
    public override string ToString()
    {
        Return $"{base.ToString()} " +
            $"I am {this.position} teacher, " +
            $"my salary is {this.success:F2) lv.";
    }
}
```

Figure 6. Task 4 - class Teacher

Teachers and students are described in a *class Staff*, which has only one field- a list of type ***Person***.

```
class Staff
{
    // School staff
    private List<Person> people = new List<Person>();

    public List<Person> People
    {
        get { return people; }
        set { this.people = value; }
    }
    public void Print()
    {
        Console.WriteLine("The Staff members are:\n");
        foreach (var p in this.people)
        {
            Console.WriteLine(p.ToString());
        }
    }
    public List<Student> FilterStudents()
    {
        List<Student> result = new List<Student>();
        foreach (var item in this.People)
        {
            if (item is Student)
            {
                result.Add((Student)item);
            }
        }
        return result;
    }
    public List<Teacher> FilterTeachers()
    {
        List<Teacher> result = new List<Teacher>();
        foreach (var item in this.People)
        {
            if (item is Teacher)
            {
                result.Add((Teacher)item);
            }
        }
        return result;
    }

        public override string ToString()
    {
        string t = "The staff members are:\n";
        foreach (var p in this.People)
        {
            t += p.ToString();
        }
        return t;
    }
}
```

Figure 7. Task 4 - class Staff

The class has two main functionalities ***FilterStudents()*** and ***FilterTeachers()***, which realize the object filtering from the both subtypes (***Student*** and ***Teacher***) and return an object list from a given subtype (*Figure 7*).

Some students use the method ***Print()*** and others use the method ***ToString()*** to export data from the collection ***Staff***.

Using FPS, provided by the LINQ library, data is inserted in the main program. By the application of *switch* and *if*-constructions, objects from the both subtypes can be created. If the data validation is successful, then the objects can be added to the collection ***Staff***. Otherwise, a message for an exception occurs in a given hierarchy setter.

It should be mentioned that some students suggest an alternative FPS for exporting the teachers' and students' lists.

```
//print all staff
staff.Print();
//alternatively print all staff
Console.WriteLine(staff.ToString());
//Filter and print Students
Console.WriteLine("Students:");
```

```
staff.FilterStudents().ForEach(st =>
Console.WriteLine(st.ToString()));
//Filter and print Teachers
Console.WriteLine("Teachers:");
staff.FilterTeachers().ForEach(t =>
Console.WriteLine(t.ToString()));
//alternatively filter and print students
Console.WriteLine("Students:");
foreach (var st in staff.FilterStudents())
{
  Console.WriteLine(st.ToString());
}
// alternatively filter and print Teachers
Console.WriteLine("Teachers:");
foreach (var t in staff.FilterTeachers())
{
   Console.WriteLine(t.ToString());
}
```

Figure 8. Task 4 – printing collection

It is observed that the students can deal with the different parts of the task to a different degree. For this reason, the students prefer to look at the solutions of the previous tasks. In this way, they assimilate more efficiently the algorithms for finding the task solutions, learning how to analyze and synthesize when they solve practical problems and realizing the gained knowledge at a higher level of abstraction. Finally, they apply the knowledge, skills and competencies acquired from module 2 and the first three tasks from the system to solve the last task. The time needed for finding the right solution of the task is different for each student and it can be an indicator for the degree of acquiring competency.

The results from solving the tasks are presented in *Table 1* and ordered by their degree of completeness. The values show what part of the ten students in the group achieved a given result.

Table 1. Achieved results

Degree of task completeness	***Task 1***	***Task 2***	***Task 3***	***Task 4***
100%	9/10	8/10	9/10	6/10
80%	1/10	1/10		3/10
60%		1/10	1/10	1/10

7. Conclusion

The suggested spiral approach is suitable to be used in the education for the occupation Application programmer, because approximately the half of the education modules are studied in two parts (IP and P, IOOP and OOP, IADS and ADS). Thus, the acquired knowledge, skills and competencies can be applied in the next education modules and upgraded spirally by giving tasks with a gradual increase in difficulty. A special feature of the spiral approach is the dependence of the degree of competency acquirement on the degree of competency acquirement from the previous task. All competencies developed by the students during the both courses are demonstrated in the last task with the exception of the competency needed for using basic design templates. In addition, the competency in editing a foreign code, i.e., finding code errors and refactoring of a code based on the criteria for writing a high-quality programming code, is also developed.

Acknowledgements

The work is funded by the MU21-FMI-004 and MU21-FMI-011 projects of the Fund "Scientific research" at the University of Plovdiv "Paisii Hilendarski".

References

[1]. White, R. (1959). Motivation reconsidered: The concept of competence. *Psychological Review*, *66*(5), 297-333. https://doi.org/10.1037/h0040934

[2]. McClelland, D. C. (1973). Testing for competence rather than for" intelligence". *American psychologist, The*, *28*(1), 1-14. https://doi.org/10.1037/h0034092

[3]. Boyatzis, R. E. (1982). *The competent manager: A model for effective performance*. John Wiley & Sons.

[4]. Raven, J. (1984) *Competence in modern society: Its identification, development and release*. London: H. K. Lewis.

[5]. Staribratov, I. (2020). Алтернативен начин за професионално образование. *Професионално образование*, *22*(2), 173-178.

[6]. Yovcheva, B. B. (2008, July). Spiral teaching of programming to 10–11 year-old pupils after passed first training (based on the language C++). In *International Conference on Informatics in Secondary Schools-Evolution and Perspectives* (pp. 171-179). Springer, Berlin, Heidelberg.

[7]. Hristov, Hr. (2011). Difficulties and Solutions when Changing the Paradigm. Teaching of Object-Oriented Analysis, Design and Programming. *Proceedings of the International Conference "Interaction Theory - Practice: Key Problems and Solutions"*, Vol. III, Burgas, 303-310. [in Bulgarian].

[8]. Hristov, H. (2010). Review and outlooks of the means for visualization of syntax semantics and source code. Procedural and object oriented paradigm – differences. *Anniversary International Conference REMIA 2010*, Plovdiv, 443-450, ISBN 978-954-423-648-9.