THE PREDICATE TRANSFORMER AND ITS APPLICATION IN INTRODUCTION TO PROGRAMMING COURSES

Prof. Magdalina Todorova, PhD
Faculty of Mathematics and Informatics, Sofia University

Prof. Daniela Orozova, PhD
Faculty of Computer Science and Engineering
Burgas Free University

Abstract: *Current article is dedicated to sharing the authors' experience in applying the predicate transformer in synthesizing (extaraction) totally correct programmes in introduction to programming courses. The training was delivered in two Bulgarian universities: Sofia University „St Kliment Ohridski" and Burgas Free University. A brief overview of known approaches to programme verification is presented, in addition some problems are analyzed and suggestions for improving the results of education in programming through using formal methods are discussed. The method for programme synthesis under discussion is based on a special function called weakest precondition. It was adapted according to the goals of education in programming based on C++. Methodologies of verification and synthesis of operators for condition and for cycle (while) are formulated. An example is used to show the application of the defined methodology, as well as the use of some techniques for defining the loop invariant. What is argued is the use of project-based approach in this education. Analysis of this approach is presented.*

Keywords: *programme verification, pragramme synthesis, predicate transformer, education in programming*

1. Motivation

The courses in programming are compulsory for all students of informatics in their first year of education. The main goal of the courses is to develop algorithmic thinking in the students, as well as to give them the foundation in structural and object-oriented programming. This knowledge are a prerequisite for studying disciplines such as: data structures, data bases, operation systems, software architecture, etc. in the next years of their education.

Another, no less important, goal of education in informatics is developing students' skills and habits related to the processes of software development, support and optimization, as well as these related to evaluation of software reliability and correctness. The degree to which these skills are developed is of crucial importance and is the real assessment which the software specialist job market gives regarding the quality of education. The high degree of integration of the information and communication technologies in almost all applications led to a need of realization of reliable programming and apparatus tools. Some of the errors in the software could prove to be of little importance in a sense. However, system mistakes which are critical to security, such as air

traffic control systems, power plant control systems, medical equipment control systems, are not acceptable. Some impressive examples of the consequences from such mistakes are well known. For example, an error in a command for division of numbers with floating point in the Intel Pentium processor caused losses estimated at about 500 million dollars. The crash of the rocket Ariane-5, which is attributed to an error in the flight control programme, caused losses of more than 370 million dollar. Due to a simple mistake, again, the medical accelerator Therac 25 caused 6 deaths (Adzhiev, 1998). The importance of programming code verification makes it a very attractive environment for training students through adapting scientific research tasks for learning purposes.

Contemporary research shows that the worldwide dominant theoretic-methodological paradigm of practical education is the constructivism, with its various theoretical branches and their applications in practice. Constructivism is based upon the idea for creating and recreating existing cognitive constructs (schemata) in the individual through the process of gaining new experience, knowledge and activity, and the process of adapting to the changing reality. Learning, regardless of the domain in which it is initiated and realized (cognitive, affective, psychomotor and interpersonal), includes a process of individual transformation (Peytcheva-Forsyth, 2010). According to the constructivists, people learn by "incorporating and integrating" the new knowledge within the existing structures of knowledge.

This article analyses the pedagogic efficiency of constructivism in the context of education in programming through using formal methods of software verification.

The following software verification approaches can be found in the literature: review (inspection), static code analysis, formal methods, dynamic methods, synthetic methods.

In order to check the programme code correctness, testing (a dynamic method of verification) is the most commonly applied in education in programming. Methods of formal verification are also used, however more rarely. The formal verification, in contrast to the other methods of verification, is based on the mathematical proof of programme correctness. In comparison to the other methods of verification, it is the most effective and reliable verification method, whose drawback is that it requires significant effort and qualified specialists in order to be applied.

In the last years, proving programme correctness has become of significant importance for the informatics science. The programming process consists of writing programmes, annotating them by preconditions and postconditions, which define the input/output specification of the programmes and prove their correctness.

However, all popular methods of programme formal verification are laborious. This justifies the need programmes to be *synthesized (extracted from their formal specification)*, i.e. to be constructed in parallel with the proof that they are correct.

The authors of this article have been applying different formal methods of programme verification in introductory courses in programming for more than 10 years. Some results were shared in (Todorova and Armyanov, 2012; Todorova, 2013). A technique for formal programme verification during programme execution, as well as its introduction in education in programming, is presented in (Todorova and Armyanov, 2012). In (Todorova, 2013), the application of axiomatic semantics and the techniques: design by contract, class invariant, proving theorems and consistency check is shown, as applied in the courses Introduction to Programming, Object Oriented Programming and Data Structures and Programming.

Current article is dedicated to applying predicate transformer in synthesizing Algol-like programmes, and programmes in C++ in particular, in the course Introduction to Programming. The choice of this method is justified by the knowledge the students possess at the end of the first term of their education: Bachelor's Degree students of the specialties Computer Science and Software Engineering.

A flaw in education to a great extent is the isolation of the courses from one another, and the lack of a single guiding strategy in structuring the learning content. The article motivates the choice of the Project Based Approach in teaching disciplines in the area, which allows for combining more requirements and knowledge, acquired in previous or parallel courses.

2. Predicate transformer and its applying in imperative programme synthesis

The synthesis method we use is based on a special function called predicate transformer.

2.1. *Predicate transformer definition and semantics*

For educational purposes, we use a predicate transformer known as weakest preconditions, introduced by Dijkstra (Dijkstra, 1975).

Definition 1. Let S is an operator, and R is a predicate, which describes the result anticipated from the execution of the operator S. ***The predicate transformer*** for S and R is the predicate Wp(S, R), which represents the set of all states such that execution of S started in any one of them is guaranteed to terminate in a finite amount of time in a state satisfying R.

Let R is a predicate. What follows are definitions of Wp for the operators of C++: empty, block, operator for assigning, operator for condition (full and short form), and operator for cycle *while*.

Definition 2. Wp(empty operator, R) = R.

Definition 3. $Wp(x = e, R) = domain(e) \wedge R(x \leftarrow e)$, where domain(e) is a predicate, describing the set of all states in which the expression *e* is defined, and $R(x \leftarrow e)$ is the predicate R where free occurrences of x are replaced by *e*.

Definition 4. Wp(S1;S2;...;Sn, R) = Wp(S1,Wp(S2; ...; Sn, R)), where S1, S2, ..., Sn are operators belonging to the subset of C++ under consideration.

Definition 5. Wp({S1; S2;...; Sn}, R) = Wp(S1; S2;...; Sn, R), where S1, S2,..., Sn are operators belonging to the subset of C++ under consideration.

Definition 6. $Wp(\text{if(B) S1;else S2}, R) = domain(B) \wedge$

$$(B \Rightarrow Wp(S1, R)) \wedge (\neg B \Rightarrow Wp(S2, R)).$$

In particular:

$$Wp(\text{if(B)S}, R) = domain(B) \wedge (B \Rightarrow Wp(S, R)) \wedge (\neg B \Rightarrow R).$$

It is often not necessary a predicate transformer Wp(if(B)S1;else S2, R) or Wp(if(B)S, R) to be found, but only to check whether the implication Q $\Rightarrow$ Wp(if(B) S1; else S2, R) or Q $\Rightarrow$ Wp(if (B) S, R) holds. The following theorem is useful in these cases:

Theorem 1. Let for the operator

if (B) S1; else S2

and the predicates Q and R the following conditions are true:

a) Q $\Rightarrow$ domain(B)

b) Q $\wedge$ B $\Rightarrow$ Wp(S1, R)

c) Q $\wedge$ $\neg$B $\Rightarrow$ Wp(S2, R).

Then (and only then) it is true:

Q $\Rightarrow$ Wp(if (B) S1; else S2, R).

Consequence. Let for the operator

if (B) S;

and the predicates Q and R the following conditions are true:

a) Q $\Rightarrow$ domain(B)
b) Q $\wedge$ B $\Rightarrow$ Wp(S, R)
c) Q $\wedge$ $\neg$B $\Rightarrow$ R.

Then (and only then) it is true: Q $\Rightarrow$ Wp(if (B) S, R).

As it is not easy in practice to use the definition of predicate transformer for the operator *while*, we formulated a theorem, through which the truthfulness of the implication can be checked

Q $\Rightarrow$ Wp(while (B) S, R).

To this end we connect with the operator while (B) S:

a) loop invariant P – predicate, which is true before and after each iteration of a loop;

b) bound function t – an integer function, which is the upper limit of the number of iteration left to be performed. The function t must be bounded below by 0 and to decline by 1 at each iteration of the loop execution.

Theorem 2. Let for the predicate P and the integer function t the following conditions hold:

a) P $\wedge$ B $\Rightarrow$ Wp(S, P)
b) P $\wedge$ B $\Rightarrow$ t > 0
c) P $\wedge$ B $\Rightarrow$ Wp(t1 = t; S, t<t1),

where t1 is the new identifier. Then the following condition holds:

P $\Rightarrow$ Wp(while (B) S, P $\wedge$ $\neg$B).

2.2. Method for programme synthesis through using predicate transformer

The method is adaptation of the one described in (Gries, 1981). It is adjusted to programme synthesis based on subset of C++. Following Theorem 1 and its consequence, we define the next methodology of synthesis of *if/else* and *if* operators.

***Methodology of synthesis of*:**

a) the operator if/else

1. Find condition B, operators S1 and S2 so the following implications to hold:

   $Q \wedge B \Rightarrow Wp(S1, R)$
   $Q \wedge \neg B \Rightarrow Wp(S2, R)$

2. Check is the following holds: $Q \Rightarrow$ domain(B).

b) the operator if – short form

1. Find condition B and operator S so the following implications to hold:
   $Q \wedge B \Rightarrow Wp(S, R)$
   $Q \wedge \neg B \Rightarrow R$
2. Check is the following holds: $Q \Rightarrow$ domain(B).

If requirement 2) does not hold, make changes in the conditions and the operators found, so 2) to hold.

As a consequence of Theorem 2, a list can be formulated, consisting of conditions for verifications and synthesis of the operator *while*.
Let the operator *while* is given, appropriately annotated with a precondition, invariant, bound function and postcondition:

```
{Q: precondition}
{P: invariant}
{t: bound function}
    while (B) S;
{R: postcondition}
```

List of conditions for verification of while-loop

1) P holds before the operator for cycle, i.e. either $Q \Rightarrow P$ holds, or an operator S0 exists so that $Q \Rightarrow Wp(S0, P)$ holds.

2) $P \wedge B \Rightarrow Wp(S, P)$, i.e. P is a loop invariant.

3) $P \wedge \neg B \Rightarrow R$, i.e. the postcondition holds at the moment of ending the execution of the operator for cycle.

4) $P \wedge B \Rightarrow t > 0$, i.e. t is bounded below by 0 as long as execution of the loop has not terminated.

5) $P \wedge B \Rightarrow Wp(t1 = t; S, t<t1)$, i.e. each iteration of the loop leads to a strict decrease of the bounding function t.

Based on this list of verification conditions, we describe the following methodology of synthesis of a programme fragment containing the operator for cycle *while*.

Methodology of synthesis of a programme fragment containing the operator while

1. Check if $Q \Rightarrow P$ holds. If it does not, search for an operator S0 so the implication $Q \Rightarrow Wp(S0, P)$ to hold.
2. Find condition B so $P \wedge \neg B \Rightarrow R$ to hold.
3. Check if $P \wedge B \Rightarrow t > 0$ holds for the found condition B. If the condition does not hold, change *t* so that the condition to be fulfilled.
4. Find operator S so that the following conditions to hold:

$$P \wedge B \Rightarrow Wp(t1 = t; S, t<t1)$$
$$P \wedge B \Rightarrow Wp(S, P).$$

One of the most difficult aspects of the method is the choice of a loop invariant. Different approaches for its construction exist. The most commonly used are: deleting a conjunct, replacing a constant by a variable, combining pre- and postconditions. The following example is selected in order to allow for easy synthesis larger amount of programme fragments based on a given formal input/output specification by using some of the above described techniques for invariant choice.

2.3. Example

An integer is given: *n*, $n \geq 0$. Synthesize a programme fragment in C++, which finds the largest integer *a*, whose square is not bigger than *n*.

Following the task, we define a precondition Q and postcondition R:

Q: $n \geq 0$

R: $a \geq 0 \wedge a^2 \leq n \wedge n < (a+1)^2$.

I solution:

If the third conjunctive member of R is deleted, the result is the following possible invariant:

P: $a \geq 0 \wedge a^2 \leq n$

We choose a bound function:

t: $n-a^2$.

After performing the steps of the methodology of synthesis of the operator *while*, the following fragment is the result:

First synthesized programme

```
a = 0;
while (n >= (a+1)*(a+1)) a = a+1;
```

II solution:

If the second conjunctive member of R is deleted, the result is the following possible invariant:

P: $a \geq 0 \wedge n < (a+1)^2$

We choose a bound function:

t: $(a+1)^2-n$.

After performing the steps of the methodology of synthesis of the operator *while*, the following fragment is the result:

Second synthesized programme

```
a = n;
while (n<a*a) a=a-1;
```

III solution:

The invariant P is found by replacing a+1 from R by the variable b and the limits of b are stated, i.e.:

P: $a \geq 0 \wedge a^2 \leq n < b^2 \wedge a < b \leq n+1$

We choose a bound function:

t: b-a-1.

Following the methodology for synthesis of the operator *while* for S0 and B, we find:

S0: a = 0; b = n+1;

B: b != a+1

In this case, we search for the body of the cycle S in the following form:

S: a = g1(n, a, b); b = g2(n, a, b);

where g1 and g2 are integer functions.
The condition $P \wedge B \Rightarrow Wp(t1 = t; S, t<t1)$ results in the following possible choices of the functions g1 and g2:

a) g1(n, a, b) = a+1; g2(n, a, b) = b, the respective body of the cycle can be in the following form S1: a=a+1;

b) g1(n, a, b) = a; g2(n, a, b) = b-1, the respective body of the cycle can be in the following form S2: b=b-1;

c) g1(n, a, b) = (a+b)/2; g2(n, a, b) = b, the respective body of the cycle can be in the following form S3: a = (a+b)/2;

d) g1(n, a, b) = a; g2(n, a, b) = (a+b)/2, the respective body of the cycle can be in the following form S4: b = (a+b)/2;

None of the found operators S1, S2, S3 and S4 can be used for a body for the cycle as the condition $P \wedge B \Rightarrow Wp(S, P)$ does not hold for any of them. We try to synthesize *if* operator, which to serve a body of the operator for cycle. As the following implications are true:

$P \wedge B \wedge (a+1)^2 \leq n \Rightarrow Wp(S1, P)$
$P \wedge B \wedge (a+1)^2 > n \Rightarrow Wp(S2, P)$

when applying Theorem 1, the next implication holds:

$P \wedge B \Rightarrow Wp(S, P)$,

where

S: if ((a+1)*(a+1) <= n) a=a+1; else b=b-1;

Thus the following programme segment is the result:

Third synthesized programme

```
a=0; b=n+1;
while (b!=a+1)
    if ((a+1)*(a+1)<=n) a=a+1;
    else b=b-1;
```

As the following implications are true:

$P \wedge B \wedge n<(b-1)^2 \Rightarrow Wp(S2, P)$
$P \wedge B \wedge (b-1)^2 \leq n \Rightarrow Wp(S1, P)$

when applying Theorem 1, the next implication holds:

$P \wedge B \Rightarrow Wp(S, P)$,

where

S: if (n < (b-1)*(b-1)) b=b-1; else a =a+1;

Thus the following programme segment is the result:

Forth synthesized progarmme

```
a=0; b=n+1;
while (b!=a+1)
  if (n<(b-1)*(b-1)) b=b-1;
  else a=a+1;
```

Lastly, as the following implications are true:

$P \wedge B \wedge ((a+b)/2)^2 \leq n \Rightarrow Wp(S3, P)$
$P \wedge B \wedge n < ((a+b)/2)^2 \Rightarrow Wp(S4, P)$

when applying Theorem 1, the next implication holds:

$P \wedge B \Rightarrow Wp(S, P)$,

where

S: if ((a+b)/2*(a+b)/2<=n) a=(a+b)/2;
else b=(a+b)/2;

Thus the following programme segment is the result:

```
a=0; b=n+1;
while (a+1 != b)
    if (((a+b)/2)*((a+b)/2)<=n) a=(a+b)/2;
    else b=(a+b)/2;
```

which can be simplified to:

Fifth synthesized programme

```
a=0; b=n+1;
while (b != a+1)
 { int x=(a+b)/2;
   if (x*x<=n) a=x;
   else b=x;
 }
```

It can be seen from this description that, in order to master and apply the method of programme synthesis, the students are required to have a very good knowledge of mathematics and basic knowledge of predicate calculus. A prerequisite for meeting this requirement is the parallel education in different mathematics disciplines: Algebra, Geometry, Discrete Structures, and Analysis. Support of the training in the field could also be secured by early introducing the terminology: precondition, postcondition, loop invariant and bound function. From the very first lectures of the course Introduction to Programming, the students are motivated to annotate all programme fragments by appropriate formal specifications; and to check if these specifications hold during programme execution as part of their work during the lab sessions.

3. Realization of the training and analysis of the results

Taking into consideration the difficult subject area and the need for applying knowledge from different disciplines, as well as from practice, an appropriate educational approach is project-based, which ensures high results in these cases.

Project-based learning (PBL) is a pedagogic model of interdisciplinary activities, related to real-life problems. This is a challenge for the learners to construct and aquire high-level knowledge and skills. The educational goals are related on the one hand to the project field, on the other hand: to developing skills for working on a project (Orozova, 2008). The basic skills to be developed by the students are: identifying the stages of project development; activity planning; keeping deadlines; collaborating with other team members; evaluating the contribution of the team members; self-evaluating; discussing on the project area and formulating and argumentatively defending own ideas and skills.

What can be achieved through team working on projects are:

- a closer connection between education and practical needs;
- enhanced cognitive activity, required of the student;
- evaluation of the developed practical skills, where the marking defines the development towards achieving the project goals;
- overcoming the difficulties encountered in collaborative working.

Having presented the theoretical basics and having given examples of application of the method of predicate transformer in synthesis of programmes using different primitive and composite structures, taught in the introductory course in programming, the students are given research tasks related to its application. The lab sessions on the discipline are dedicated to giving explanations on: the educational method used – PBL, the learning environment Wiki; the teams are formed (usually of two students).

An important part of PBL is project preparation and planning. The activities are related to formulating the tasks, defining the stages of work, the sub-tasks, deadlines, information resources, milestones, etc.

The main activities of the students performing project work, which were applied during the training, could be identified as (Tuparova and Tuparov, 2010):

- defining the tasks and discussing the topics for each group (team);
- defining the different subtasks and assigning them to particular students in the team;
- defining the timetable for execution;
- defining the milestones and the particular artifacts which must be produced at each stage;
- defining the form of the final project results;
- presenting the project, related to: presenting the tasks, which were completed to achieve the goals; description of the results; answering questions related to the project topic and the tools used for working on the project.
- documenting, related to: technical parameters of the task, work plan, interim results and discussions;
- reporting the results: preparing a presentation, presenting text or table data, presenting the project to an audience.

Another important side of PBL is that the assessment criteria and means must be defined and announced. The criteria we used for the training under consideration were:

- defining what is to be assessed: goals achievement, keeping deadlines, quality of the product, etc.
- defining what means to be used to assess: formative assessment (observation, check-lists), summative assessment and marking of the final products, the work of the team and each of its members, project presentation and defense.

In addition, environment must be provided for the students to share opinions and evaluation regarding the quality and characteristics of the achieved solutions, using different technologies. The students should be given the opportunity to compare and evaluate the results achieved by the other teams.

In the course under investigation, using the approach mentioned above, we use Wiki environment for public and absolutely transparent environment for documenting the process of project development, as well as for communication among the participants. Communication is Wiki is at three levels: between the lecturer and all students, among the team members, and among the teams. The coordination and communication at the team level are of crucial importance. The philosophy and technology behind Wiki ensure the users to have relative equality and autonomy in the process of working, and the communication is „horizontal", of the kind many-to-many. This feature totally reflects the contemporary understanding of interactive and collaborative learning, which is focused on the learner, while the lecturer takes the new role of facilitator and mediator (Atanassova

and Orozova, 2011). Saving all versions of each page in the form of detailed history and its availability at all times allows the contribution of each student to be identified.

Taking into account the requirements for project documentation design, formulated by experiences methodologists, we defined the following elements for the documentation of each project:

- a timetable agreed among the team members, which corresponds to the activity timing and respective person; the timetable should make it clear which are the key moments and relations between the activities (for example, a diagram of the sequence could be used);
- team discussions on the given task and the problems it entails, including discussing different approaches for solving them and relevant information sources;
- personal notes of the different members on the respective activities;
- a diary with the concrete individual activities with justification on each change of the project condition;
- individual reports, stage reports and a common (team) report on the given tasks;
- collection of correctly cited resources (literature, internet addresses, developer's and users' manuals, etc.), which are used by the students to justify their decisions.

The most important part of the project work is finding a correct solution to the given problem. In the particular case, the students must find a solution to tasks related synthesis of C++ programmes. Finding solution goes through four main steps (stages), visualized on fig. 1:

(1) Define the input/output specification, which the synthesized programme must satisfy. In case the task presupposes a cyclic process, an invariant and bound function for the operator of the cycle are defined.

(2) Apply the methodology for synthesis of a progarmme segment, containing the operator *while*. In case the implication $Q \Rightarrow P$ does not hold, first check if S0 can be assignment operator so that $Q \Rightarrow Wp(S0, P)$ to hold. If this is not possible, the methodology for synthesis of an *if*-operator is applied and S0 in the form of an *if*-operator is searched for. Defining the operator S, giving the body of the operator of cycle, begins by checking if S can be assignment operator, then – *if*-operator.

(3) Try to find more solutions of the task. In order to do this, use so far unapplied techniques for defining cycle invariant: deleting a conjunct, replacing a constant by a variable, combining pre- and postconditions, etc.

(4) Using C++ programming environment, additional verification checks of the synthesized programmes correctness should be performed. This stage is to restate our belief that a perfect verification method does not exist.

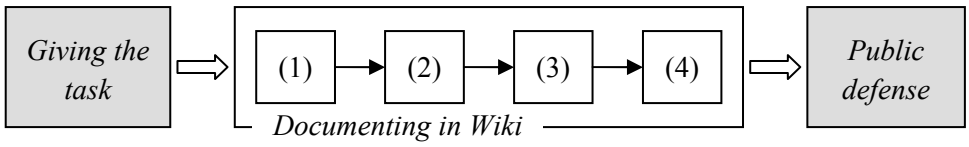

Fig. 1. Working along the project stages

The analysis of the conducted training was performed at three stages: immediately after the project defenses; after the exam on Introduction to Programming, and after the students realization as software specialists. The analysis is based on students' achievements, their activity during the lectures and lab sessions, and on communication between lecturers and students via various channels (consultancy, forum, e-mail).

The data collected are used for statistical analysis of students' skills. What is assessed is the knowledge and errors the students make, with the focus on the latter during the next stages of education.

The data analysis shows that, despite the difficulties which appear during the work, the application of the method of predicate transformer for the purposes of education in informatics is realistic and efficient. We found with satisfaction that, during the training, the students are enthusiastic about searching for non-traditional and original ways of solving the tasks, especially at the stages of defining the specification, as well as about project development and documentation. It is often during the work that students additionally encounter and use scientific results, which are not presented as a learning material in the current courses.

Applying Project-Based Learning also met our expectations for effectiveness. Its use stimulates students' interest and made their project work close to scientific research.

Although the goals were completely fulfilled primarily by students with good background in both programming and mathematics, we argue that the suggested approach is relevant to a great extent for this type of training. As a result of the training, students awareness was raised regarding the application of methods of formal verification (and in particular the method of synthesis of C++ programmes) and the benefits they give. They also got the awareness that programming is a serious scientific activity. In addition, they realized the importance of the activities related to verification of programme code. Furthermore, they developed understanding about the usefulness of mathematical specifications for the process of programming, as well as for the design of the programme code. Students reached the understanding that mathematic knowledge is needed in programme development.

We also registered raised motivation in the students with lower levels in mathematics towards higher efforts to improve in this area. Secondary goals, such as identification and development of team working skills, were also achieved. Though the process of solving problems together, the students gained skills in collaboration.

After completing the course, we ran an additional test in order to identify the degree of knowledge retention and to compare with the project assessment. Both the test and the project can have a maximum of 20 points. The results are gathered in three groups (from 8 to 12 points, from 12 to 16 points, and from 16 to 20 points) and are presented in fig. 2.

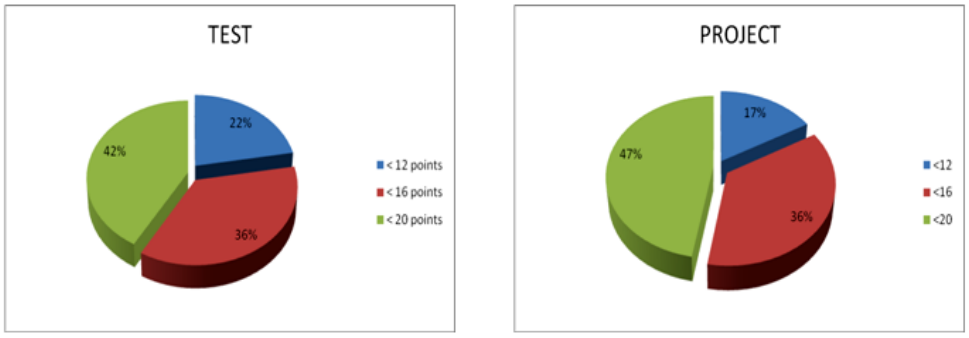

Fig. 2. Results of the test and the project

The analysis of the results of the test, the project and the final assessment of the discipline Introduction to Programming show a tendency towards increasing students' degree of knowledge gain in comparison to these trained without formal methods. Questionnaires were administered at the end of the training, which aimed at researching on the students' interest towards applying formal methods of verification and synthesis of programmes in the introductory programming courses. The results show that, despite the difficulties, the students are motivated to study and apply such methods; and that using PBL in such courses is a prerequisite for active learning. Some of the students who were trained via the method of predicate transformer for programme verification and synthesis were motivated to research the area and defended theses in the field of programme code synthesis (Trifonov, 2012; Nikolov, 2013).

4. Conclusion

The main conclusion made by the authors is that the education in programming must follow the requirements of the software market, as well as the development in the information and communication technologies. The latter are rapidly introduced in the everyday life and become the foundation of contemporary society. Considering this, special attention should be given to introducing the most effective and reliable methods and techniques for programme code verification, which to follow this dynamics.

The authors have been working with the belief to make a step ahead in this direction. The results of the education motivate further research on adapting techniques and application for formal verification in education.

Project-Based Learning used as a learning method in this context supports not only to increase the effectiveness, but also to create software professionals of the students. Working on a project the students have to: study the given task; collect and analyze data from different resources; share, generate and discuss different ideas; make own justified suggestions, hypotheses and predictions; conduct and analyze own experiments; create artifacts (reports, data bases, multimedia, prototypes, etc.); create proofs, make summaries and conclusions; report and present their ideas and findings in public; identify new problems and questions. Together with the "hard" skills connected to the particular discipline, the learners also develop "soft" social skills, which are manifested by more responsibility regarding the personal contribution in the team work, and improved communication skills and coordination within the team.

References:

1. Adzhiev, V. (1998). Safeware Myths: Lessons of Famous Catastrophes, Open Systems Journal, Moscow, 6(32), 21-30 (in Russian).
2. Atanassova, V., Orozova, D. (2011). Project-based Learning on Databases in a Wiki, International Conference of Free University of Burgas, 279-286 (in Bulgarian).
3. Dijkstra, E. W. (1975). Guarded commands, nondeterminacy and formal derivation of programs, Communications of the ACM, vol. 18, Issue 8, 453-457.
4. Gries, D. (1981). The Science of Programming, Springer-Verlag, New York.
5. Nikolov, K. (2013). Analysis and extraction of programs (models and methods for verification of properties of non-relational databases), Sofia University, Bulgaria, Supervisor: M. Todorova.
6. Orozova, D. (2008). Possibilities of Project-based Learning, *Burgas Free University Annual*, Volume XIX, 301-305 (in Bulgarian).
7. Peytcheva-Forsyth, R. (2010). E-Learning - theory, practice, pedagogical aspects of the design, *Sofia University e-Journal*, 1, 1-7 (in Bulgarian).
8. Todorova, M., Armyanov, P. (2012). Runtime Verification of Computer Programs and its Application in Programming Education, *Global Science and Technology Forum: International Journal of Mathematics, Statistics and Operations Research*, Vol. 1, No. 1, Singapore, 105-110.
9. Todorova, M. (2013). Applying Program Verification Methods in Software Specialists Education, *Proceedings of INTED2013 Conference 4th-6th March 2013*, Valencia, Spain, 6260 – 6270.
10. Trifonov Tr. (2012). *Analysis of methods for extraction of programs from non-constructive proofs*, Ludwig-Maximilians-Universität, Münhcen, Germany, Supervisor: H. Schwichtenberg.
11. Tuparova, D., Tuparov, G. (2010). Management of students' participation in e-learning collaborative activities, *Procedia - Social and Behavioral Sciences*, Vol.2, Issue 2, 4757–4762.