# Automatic Tests and Practical Tasks Generation in Distance Learning Systems

Dmytro Kravtsov[1]

[1] Kherson State University
hwndmaster@gmail.com

**Abstract.** This article deals with the problem of the developing and implementation of the tests and practical tasks generator in the distance learning systems. The object model of the tests and exercises generator is developed based on the mathematical model.

**Keywords:** Smart-test, mathematical test, automatic tests building, remote testing, remote practical tasks.

**Key Terms:** InformationTechnology, KnowledgeEngineeringMethodology, Development, ModelBasedSoftwareDevelopmentMethodology.

## 1 Introduction

The software module that support practical tasks and academic performance ratings during the learning process are the most important in distance learning systems (DLS) [1]. A test is the major element of the academic performance ratings in DLS. Generally, tests consist of the tasks (questions) that deal with the all topics of the studied subject matter. Practical tasks and tests' questions in some sciences, more often exact sciences such as mathematics, physics, chemistry, informatics, etc., are characterized with the similar wordings that may be shown with templates. Tasks templates are the formal expressions that are described with the strict syntax and have well-defined semantics. These expressions contain parameters with the exact given range of values. A task instance is created with the substitution of the specific parameters of the defined area of values into the sample.

The practical tasks and tests development is rather hard and tedious process of tasks (questions) creation, possible answers defining, and correct answers detection [2]. The following drawbacks of the test modules are typical for the majority of the wide-used DLS:

- time consuming process of tasks (questions) compilation;
- requirement to develop a wide variety of similar tasks to provide students with representative sets of questions or to solve a problem of students copying the each others tasks solving;

– possible mistakes committed by a tasks compiler.

The good example of the software implementation for this type of tasks is Generator of examination tests in software environment «Examination tests delivery system» [3].

## 2 Problem Definition

While DLS testing module engineering, the process of testing is usually understood as a set of statements calls with a certain behavior in the context of a learning model. The process of the automatic test compiling in DLS testing module is a work flow in which a certain logics with the possibility of automation is implemented. It is important to start with a determination of an abstract model, that contains all the necessary data describing the learning aim and explaining the correlation between objects of the specific instances of this model, to automate the compiling of tests' questions as a work flow.

Let's say *smart-test* is the instance of the test, generated by the given template with the automatic test generation process that is described above.

The automatic generation of a smart-test will help to substantially simplify the practical tasks compilation in the mentioned above sciences, reducing required time and guaranteeing the unique character of each task generated, as well as it gives the ability to automate the tests results evaluation without negative human factor effect.

In this work we consider the problem of creation of a smart-test generator abstract model and tasks for solving the most of the practical tasks of the applied disciplines, and the implementation work of the smart-tests generator in DLS. The abstract generator model is a generalized model of *mathematic tests* [4]. The results of the work will be illustrates with the examples of the specific mathematics discipline tasks.

## 3 Abstract Model

We define the smart-test as a set of learning tasks set and test settings: $ST = <S(s_1, ..., s_L), P(P_1, ..., P_N)>$, where $S(s_1, ..., s_L)$ – a set of test settings, defined by a user (a test compiler), and $P(P_1, ..., P_N)$ – a set of learning tasks, that define the term *mathematic tests* [4].

To build the $P$ – abstract model of the learning task – we use its mathematical definition [4], that has the following view:

$$P = <M, \Phi, Q>,$$

where

$M$ – a set of the task input parameters, and is described as $M(x_1, ..., x_n)$, where $x_1, ..., x_n$ – the task parameters;

$\Phi$ – a set of the task conditions, described as $\Phi(\varphi_1, ..., \varphi_k)$ , where $\varphi_1, ..., \varphi_k$ – task conditions;

$Q$ – the task result model, described as $Q(q_1, ..., q_m)$, where $q_1, ..., q_m$ – sets of correlations, defining the results of the task.

This mathematic model is sufficient for further abstractions building and object-realization with the high-level programming languages.

Let's agree with the following terms to describe the mathematic model above in a programming language:

$P = <M, \Phi, Q>$ – LrnTaskTemplate (abbreviated from Learning Task Template);

$M$ – LrnTaskModel (abbreviated from Learning Task Model);

$\Phi$ – LrnTaskValidator (abbreviated from Learning Task Validator);

$Q$ – TResponse, that is the element of the software module «Solver» LrnTaskSolver<TModel, TResponse>, that will be described below.

The interrelations scheme of the objects described above is introduced at fig.1.

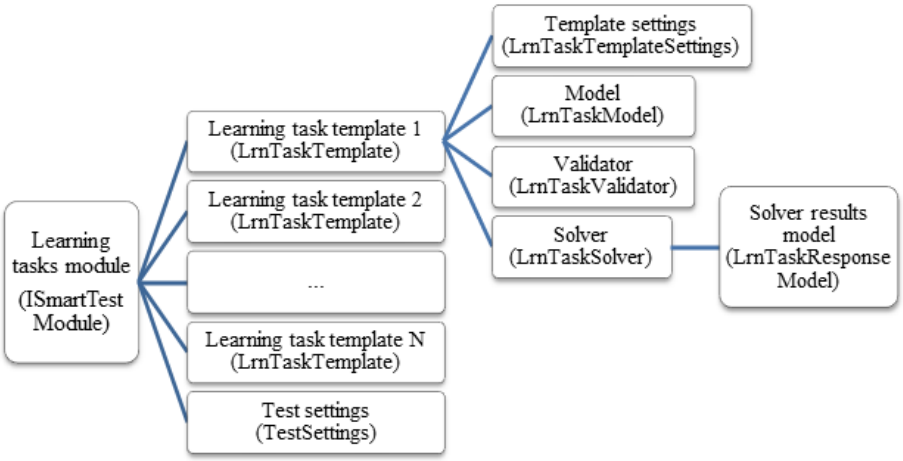


**Fig. 1.** Smart-test object model elements interrelations scheme.

On this scheme ISmartTestModule is a template library of the learning tasks generators for smart-tests compilation.

A *template library of the learning tasks generator module* is a program class (LrnTaskTemplate), which contains all the functions necessary for building the learning task instances, their correction, and solvings with the use of named model.

Task generator template functions use general (TestSettings) and local (LrnTaskTemplateSettings) settings, given by a user (test compiler) as input parameters. These settings may provide the template with generation and validation parameters of the specified learning tasks as: instruction to use preferable types of questions [7], number of answers to generate, module fine-tuning with including/excluding certain templates, field of input parameters, etc. The number of such settings may differ depending on the module and its templates. It is expected that the test compiler tune the settings while adding a smart-test to the system.

*Learning task model* (LrnTaskModel) is the program class that aim to store the input parameters of the learning task. Depending on the specified aims, the learning

task model may contain advanced parameters, necessary to describe the task settings. The number of the parameters is not limited by the system.

The next learning tasks generator element is LrnTaskValidator class. The aim of this program class is *validation* of the model instance to correspondence to the task settings. This program class is also used in reconciliation of the generated learning task model that is described further.

A *solver* (LrnTaskSolver) is the final element of the template. The aim of the solver is to solve the specified task (achieve results), using the LrnTaskModel parameters with the conditions given in LrnTaskValidator. The results of the solver are the formal expressions in the signature of the certain subject field that may be defined in the LrnTaskResponseModel class inheritor. In particular, atomic expressions may be there – a number, a string, a date.

## 4 The Model of the Smart-test Implementation Software Module

The model of the learning tasks generator template module is shown at fig. 2.

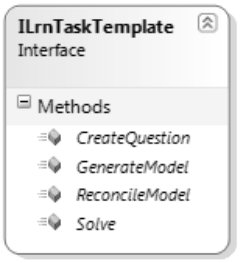


**Fig. 2.** The model of the learning tasks generator template module.

The list of the main *functions of the tasks generator* is below:

1. GenerateModel is a method of learning task model instance creation (LrnTaskModel).
2. ReconcileModel is a method of generated learning task reconciliation. This function is necessary to solve the problem, when the system generates similar tasks at the first stage of task generation in the set of the final smart-test. The correction algorithm must be implemented locally for each type of the learning tasks. The aim of the correction is to change the learning task model by making it unique comparing with other tasks in the set of the tasks in the smart-test.
3. CreateQuestion is a method of the question description creation. This method creates a program class instance that describes the question. This class model is shown at fig. 3:

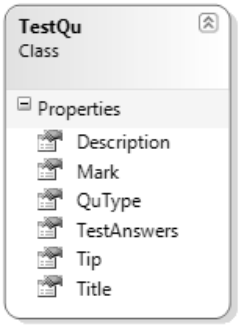


**Fig. 3.** Base model of the program class containing question description.

Here is the description of *model properties*:

- Title - title-text of the task. This property consists of the short text, that explains students the subject of the question;
- Description - the description of the task that will be shown to the student. An author of the specified learning tasks module implementation should consider possibility to create description text on one of the system languages (for example, Russian, Ukrainian, English);
- Tip - a hint text, that the student may use answering the question. This field is optional and may be left empty. Usage of the tip is fixed and processed by the system;
- QuType - type of the question. An author of the specified learning tasks module implementation may use one of the multiple questions types, mentioned in the specifications [7];
- Mark - a grade that a system assigns for the correct answer for the question.

4. Solve - a method of the task solving. This function uses the learning task model (LrnTaskModel) as the input parameter and solves it, using the functions of the LrnTaskSolver program class. The results of the solving returns as the object, consisting of the correct answer for this learning task.

## 5 Examples of Smart-tests Implementation in DLS Kherson Virtual University [5]

Here is the example of learning tasks templates implementation on “Quadratic equation” topic [4, 6]. Below you can see the objects implementation, which inherit properties and methods of the abstractions described above:

1. QuadraticEquationTestModule learning tasks module, functions and properties of the program interface ISmartTestModule (fig.4):

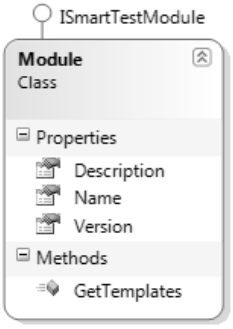


**Fig. 4.** QuadraticEquationTestModule program class.

2. Template 1 - QuadraticEquationTestTemplate1, that implements functions and properties of the program interface ILrnTaskTemplate. This template will generate tasks with the title: "Solve the quadratic equation with integer coefficients";
3. Template 2 - QuadraticEquationTestTemplate2, that implements functions and properties of the program interface ILrnTaskTemplate. This template will generate tasks with the title: "Find the sum of the squares of the roots of a given quadratic equation" (fig. 5):

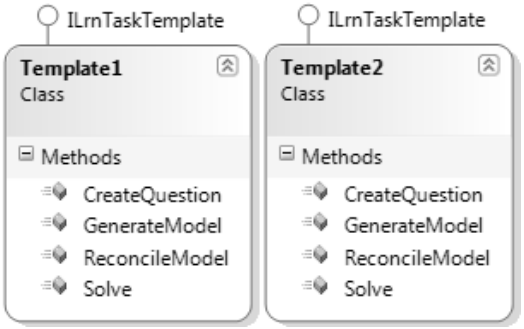


**Fig. 5.** QuadraticEquationTestTemplate1 and QuadraticEquationTestTemplate2 program classes.

4. QuadraticEquationLrnTaskModel implements the functions of the abstract program class LrnTaskModel (fig. 6). This model is actual for the both templates:

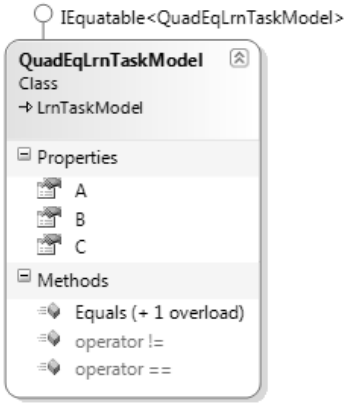


**Fig. 6.** QuadraticEquationLrnTaskModel program class.

5. QuadraticEquationLrnTaskValidator implements the functions and properties of the abstract program class LrnTaskValidator (fig. 7). This validator is actual for the both templates:

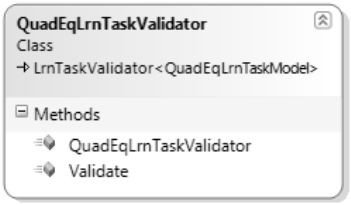


**Fig. 7.** QuadraticEquationLrnTaskValidator program class.

6. Template 1 Solver - QuadraticEquationLrnTaskSolver implements functions and properties of the abstract program class LrnTaskSolver (fig. 8):

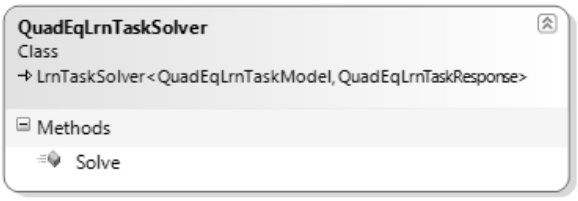


**Fig. 8.** QuadraticEquationLrnTaskSolver program class.

Executing the solution from the parameters of the mentioned model, this solver returns the object of the QuadraticEquationLrnTaskResponse type (fig. 9), that contains a pair of numerical values: $x_1$, $x_2$:

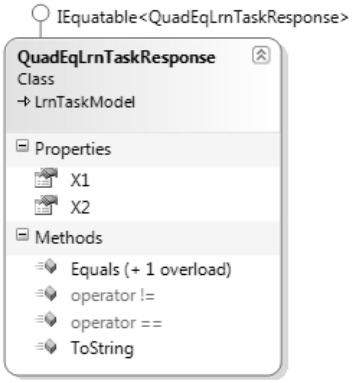


**Fig. 9.** QuadraticEquationLrnTaskResponse program class.

7. Template 2 solver - QuadraticEquationLrnTaskSolverU, which implements functions and properties of the abstract program class LrnTaskSolver (fig. 10). Executing the solution from the parameters of the mentioned model, this solver returns one numerical value x:

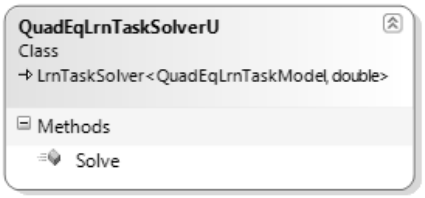


**Fig. 10.** QuadraticEquationLrnTaskSolverU program class.

8. Test setting class - QuadraticEquationTestSettings implementing the program interface ILrnTaskTemplateSettings (fig. 11):

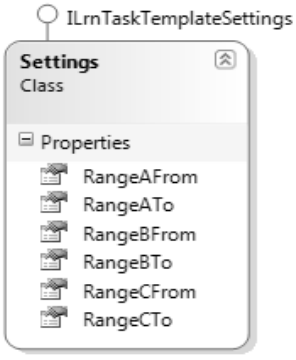


**Fig. 11.** QuadraticEquationTestSettings program class.

In this example it is supposed to specify valid values in the test settings for quadratic equation coefficients: *a*, *b* and *c*.

Now let us turn to the algorithm of creating smart-test and learning tasks using the object model, mentioned above.

The process of the smart-test creation begins with the connecting necessary modules to the system interface. We should mention that in this example the user has just one module: QuadraticEquationTest. The user should point in the module settings the templates to use (there are only two in this example – Template 1 and Template 2), and the settings of each of the template. The test compiler also sets number of questions: *N*, which should be generated. One of the parameters, that the user should specify, is the parameter of the rate set, which specifies the coefficients of usings of module templates in the smart-test.

When all the settings are made by the user, a smart-test is triggered after each request of a student in the testing group of the distance learning system. Below you can see the algorithm of the test instance generation (fig. 12).

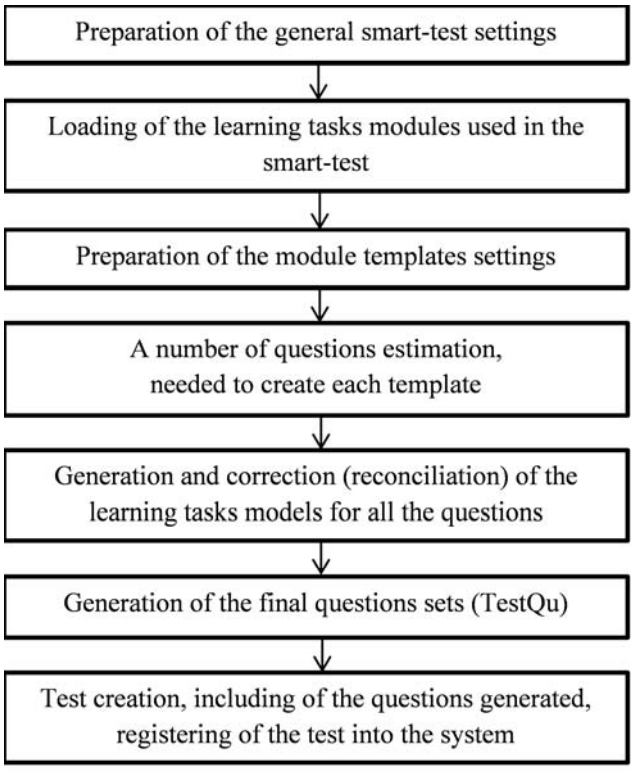


**Fig. 12.** Algorithm of the final test instance generation scheme.

**A number of questions estimation.** Assume $N$ - a number of questions, needed to generate the test, $T = <t_1, \ldots, t_k, q_1, \ldots, q_k, s_1, \ldots, s_k>$ - a set of used templates from all the connected modules, where $t_i$ – the instance of template class, $q_i$ – rate of using of $i$-template to the all number of templates $T$, $s_i$ – template settings, made by the test compiler.

**Example.** Assume $N = 20$, $T = <t_1, t_2, 0.3, 0.7, s_1, s_2>$.

Then the final test will consist of 20 questions, based on the templates $t_1$ and $t_2$. 30% of the questions (6 questions) will be generated based on the first template. 70% of the questions (14 questions) will be generated based on the second template.

**Learning tasks modules creation.** Below you can see the algorithm scheme of the learning task module creation and correction (fig. 13).

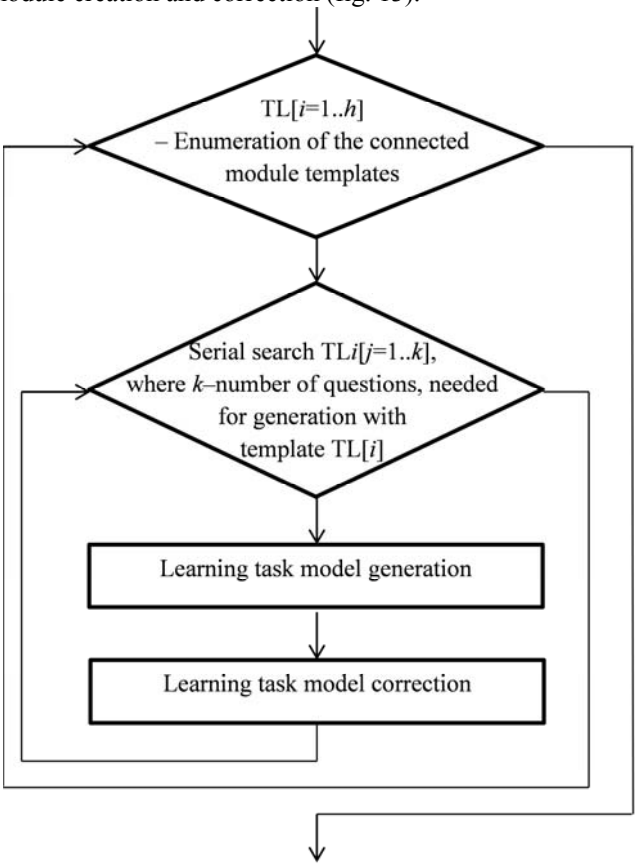


**Fig. 13.** Scheme of the sub-program algorithm of the learning task model generation and correction.

As one can see from the scheme above, model correction is implemented immediately after it is generated. It allows, first of all, using rough method of the model values generation, using the system random number generator. Then in the correction function the model parameters are changed not to coincide with the other models, generated in the cycle before.

**Generation of the final questions sets (TestQu).** The process of the final set of questions creation, based on the already generated and adjusted ones, occurs with the same enumeration as a generation of the learning tasks models - first all the templates of the connected modules are enumerated, and then the number of this template questions is searched through.

The sub-program of the learning task model question instance creation is shown at fig. 14.

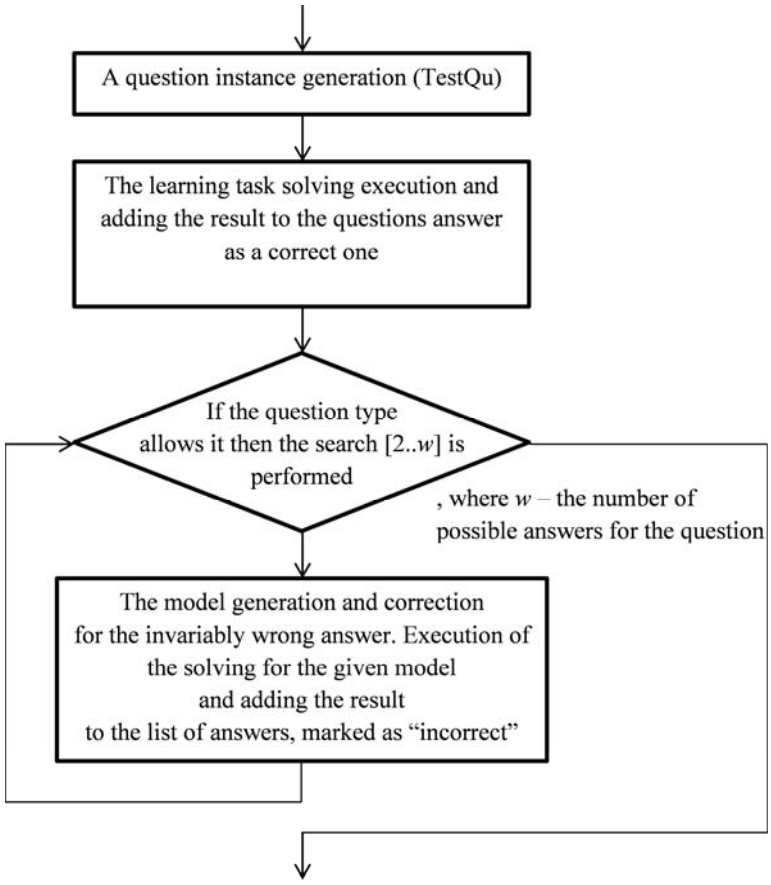


**Fig. 14.** The question instance generation sub-program algorithm scheme.

Thereby, smart-test is initializes and registers in the distance learning system, and then it becomes ready for students to pass it.

## 6 Conclusion

The structural model of the software module for the smart-tests generation and use, based on the method of automatic generation with the assigned tests or practical tasks template is build on the basis of mathematical model.

The software module classes, algorithms of learning tasks and questions models generation and verification, and smart-test answers processing are described. The described model was tried and tested on the examples of the smart-test for the exact

sciences: mathematics, physics, chemistry, computer science. Also it seems possible to make smart-tests in the field of some other disciplines: some biology sections, psychology, music, sociology, etc.

This method of automatic test generation, in the long view, may be used while creation of adaptive tests with the use of smart-tests.

## References

1. Bykov V. Yu.: Models of the Open Education Organizational Systems: Monograph. – Kyiv: Atika, (2009) – p. 684. (In Russian)
2. H. Kravtsov, D. Kravtsov.: Knowledge Control Model of Distance Learning System on IMS Standard / Innovative Techniques in Instruction Technology, E-learning, E-assessment, and Education. – Springer Science + Business Media V.B. pp.195 – 198 (2008) (In Russian)
3. Kruchinin V.V., Magazinnikov L.I., Morozova Yu.V.: Control tasks generator, VIII International scientific conference and exhibition "Unified educational information environment: problems and development trends", Tomsk, 17 (2009), http://www.ict.edu.ru/vconf/index.php?a=vconf&c=getForm&r=secDesc&d=mod&id_vconf=30&id_sec=174. (In Russian)
4. Lvov M.S.: Mathematical tests in the systems of computer mathematics of the educational purpose. / M.S. Lvov // Control systems and machines. - 2011. - №6. (In Russian)
5. Distance learning system Kherson Virtual University, http://dls.kherson.ua/dls.
6. Quadratic equation, the Free Encyclopedia, Wikipedia, http://en.wikipedia.org/wiki/Quadratic_equation.
7. IMS Question & Test Interoperability Specification, http://www.imsglobal.org/question/