UDK 614.2+574/578+004.38

O.A. Litvinov, D.L. Gruzin

FEATURES OF AUTOMATING THE PROCESS OF DEVELOPMENT OF THE FUNCTIONAL COMPONENTS OF INFORMATION SYSTEMS

Abstract. The article proposes the idea of the approach to the construction of information systems, which simplifies the creation of functional components responsible for the behavior of the system. Solution to the problem is based on the application of the hierarchy of templates that allow you to describe and re-use the experience of successful development. An important feature of the proposed solution is the simplicity and accessibility of filling database templates by ordinary user-programmer.
Keywords: automation of process, code generation, reuse.

Actuality of the topic. At the moment, the important practical problem of software developers is the rapid creation of high-quality software. Quality implies that the product will be comply with a number of characteristics (quality attributes), among which should be noted: stability, testability (testability), utility (usability), availability, scalability and openness. This imposes additional constraints on the development / rules: the templates (eg., SOLID) and styles, documentation, research, usage of multiple tools, different platforms and technologies.

In general, a modern information system can be described as a layered and heterogeneous, i.e. consisting of a number of different modules (components, subsystems), created using different tools and working on different platforms. The construction of such systems usually require a lot of different professionals and well-organized process that guarantees the quality of development [CMMI]. The most important component of this process is to analyze and use the experience of previous developments, including aspects of the planning, management, creation and implementation of technical solutions. This allows you to identify the most effective solutions those or other problems, optimize the process and minimize costs. An important feature here is that for reuse of a component, it must be a formal description. The quality of this description depends on the frequency of its use in a manual search for a solution of the problem and the ability to automate its application.

In general, the process of developing an information system is reduced to the transformation of the functional requirements of the user and the desired quality

ISSN 1562-9945

attributes of the system in a turnkey solution: a lot of software components that work in the operating environment. Functional requirements represent a formalized successful and unsuccessful (basic and alternative) scenarios of user interaction with system (action-reaction). For simplicity of conversion and tracking requirements in the solution (traceability) functional requirements are divided into levels of abstraction: business, system, component. For simplicity of planning tasks from these scenarios are pick out user stories, which include acceptance tests (user acceptance tests) and a discussion on the implementation. Necessary conditions for the start of development (formation number of tasks - backlog) should be considered as the presence of test acceptance and system architecture, which is based on the attributes of quality. By the term "architecture" is defined here as a set of standard solutions of typical tasks dictated by a set of attributes of quality and specificity of the developed system. The system architecture can also be represented at different levels of abstraction: from high-level layers to the components required in these layers (Helper, Manager, Strategy, Validator, Sender, Receiver, Processor, etc.). Formation the dependency graph of tasks (eg., The development of the structure of tables, service, DAO and DTO objects, UI components, and so on.) is the basis for development planning. Each task is transformed into a set of unit tests used to estimate the performance of the contract by developed components. A number of related components that implement the tasks checked automatically or manually using the test acceptance, their breed.

The first task of automation of building solutions is to create a set of components for a task, including the required components (interfaces, testing, implementation, exceptions). If you solve this problem - the development will reach a new level of quality, as much of the routine work to create components will assume a code generator or DSL interpreter. It should be noted the well-known problem of domain-specific languages: the effectiveness of the solution of typical problems and the complexity and inefficiency of improvements, expanding by developer.

The second task □ the transformation of formalized tests of acceptance to a set of tasks, according to the chosen architecture: according to one of the branches of the scenario, described in the language L, using previous experience, formalized in the form of «LT», the system should offer dependency graph of tasks TDG, which will be converted to full or partial set of components.

In both cases, an important issue is to teach the system to carry though such transformation, with the facility should be accessible and convenient for the developer. In this article we consider the variant of the decision for the first task.

Analysis of recent publications. Consideration of issues generating a structural component of the system is not interesting, for realization such tasks there is a wide class of tools Visual Paradigm [1], Enterprise Architect [2], which are closely related with universal modeling language UML [3]. The main issue of this article is the ability and completeness of the generation of functional components that matches to modern requirements. In the article [Gruzin 2014] was presented interpretation of the model-based approach that simplifies the creation and maintenance of an information system, which is based on the model, described using the frame approach of knowledge representation. On the basis of the described model generates a number of components, which greatly simplifies the process of developing multi-layer applications, as well as the interpretation of the model is made during system operation, that excludes the need to build and test excess classes. However, this approach is suitable only for the class of deterministic document-oriented systems: the rate on the interpretation of the results in a loss of flexibility, openness of the system is lost, extensible markup requires further means of interpretation that is not available to the end user-programmers.

Statement of the task. Searching of the flexible and easy-to-use tool to ensure maximum efficiency in the building of high-quality software modules, involving a minimum effort on the description of the task from the developer, as well as having the opportunity of learning transformation script to executable code, is the aim of this work.

The main part. In this article, we consider the aspects of automation building functional components responsible for the implementation of tasks. In general, such a component represents the next set: contract, implementation, unit tests (Fig. 1). The contract includes three components: an interface exceptions and domain. Unit tests testing the contract. Fig. 1 shows the relationship between the three components of the contract and developed component, it is worth noting that all of these components depend on the domain.

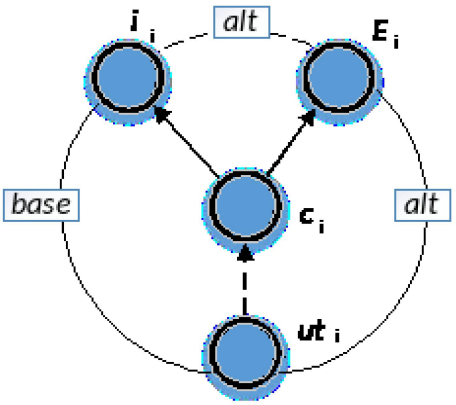

Figure 1 – The relationship between three components of the contract and developed component:
i - interface, E - number of possible exceptions, ut - unit tests, c - the component, alt - alternative scenario, base - base scenario.

An integral part of all approaches aimed at improving the quality of the development process is the education and analysis of best practices after finish of the stage or implementation of the product. Standard is processing code with detailed commentary and subsequent documentation of successful solutions.

The proposed idea is based on the extension of this approach: creating templates that describe the solution at different detail levels, which are aimed at automation of obtaining new solutions based on formal experience. The detail level should be sufficient to describe the minimum structural elements, which allows to describe the function or even part of the function, followed the class, the resource associated with the execution of tasks and is reflected in a set of classes, dictated by the architecture.

This pattern can be defined as a means of describing the display initial (domain-oriented) description given in a language that includes target and number of semantic markers that describe the transformation into a group of structures close to the executable form (files, classes, projects, scripts creation and changes in the structure of the database).

$$t_i : D_i \rightarrow R_i \, t_i \in T, D_i \subseteq D, R_i \subseteq R \quad , \qquad (1)$$

where, t – mapping of structures of description in the language D to target shape R, constructs are responsible for the implementation of a business function or part of it, the shape of which is close to the real machine executable form (here the real machine consider a number of components capable of providing the program described in the language of the target constructs).

To ensure the desired performance, and exclusion of loss of flexibility is supposed to use a plurality of language levels for describing patterns-mapping, wherein the results obtained after transformation by the higher level class can serve as input data to display a lower level.

$$\tau_i : \Delta_i \rightarrow D_j D_i, D_k \subset D_j \ . \tag{2}$$

Wherein

$$t_i : D_i \rightarrow R_i \ ,$$

$$t_k : D_k \rightarrow R_k \tag{3}$$

It is important to note that in terms of low-level architecture patterns are not differentiated. It gives independent for developers in selecting an architecture, setting new variants of the components.

Consider the example. The first mechanism in the hierarchy of templates is the transformation at the primitive-functions level (see Eq. (1)). View of template shown below, a marker of "$" marked non-terminal slots, ie, parts that can be replaced by the input data.

```
AlternativeTestMethod
[[
 2/// <summary>
        /// An alternative story of .
        /// </summary>
        [Test]
        [ExpectedException(typeof($ex))]
        public void $m()
        {
                3try
                {
                   4// Arrange
                   // Act
                   $txt;
                   // Assert

                3}
                catch (ExceptionBase ex)
                {
                        4string message = ex.GetMessage();
                        Console.WriteLine(message);
```

```
                    throw;
                3}
            2}
]]
```

Listing 1 – Template of function-class level

Task for the transformation with using this template described by next structure

<<AlternativeTestMethod(m: **Something**, ex: **SomethingIsNotFound** , txt:)>>,

bold marked the values of slots.

The result will be generated objective function (marked in Listing 3 by bold).

The transition to a higher level (level of files) aims to simplify the recording of the composition of functions combined in a module file. For this serves to introduce an additional level of templates describing the transformation in the some form (according to the formula (2)).

atstm= <<AlternativeTestMethod(m: {0} , ex: {1} , txt: {2})>>;

Using of transformation - Domain-oriented description at level 2.

```
--DealerManagerTester2.cs->
tst:-UniTrader.DealerBox.Core.Contract DealerManager %We are going to make something new.%;
atstm:-Something SomethingIsNotFound;
etst:-;
```

Listing 2 – Description of task for transformation at the level of files

The result is the target file with name DealerManagerTester2, which cimtain method Something, bold part was generated by using the transformation at the level 1.

```
namespace UniTrader.DealerBox.Core.Contract
{
    using NUnit.Framework;
        /// <summary>
    /// We are going to make something new.
    /// </summary>
    [TestFixture]
    public class DealerManager
    {
        /// <summary>
```

```
                /// An alternative story of.
                /// </summary>
                [Test]
                [ExpectedException(typeof(SomethingIsNotFound))]
                public void Something()
                {
                        try
                        {
                        }
                        catch (ExceptionBase ex)
                        {
                                string message = ex.GetMessage();
                                Console.WriteLine(message);
                                throw;
                        }
                }
        }
}
```

Listing 3 – Result of the transformation

The next level is the level of resources. It seek for describe a component responsible for implementing the contract (fig. 1).

```
!resource
resource DealerResolutionManager in UniTrader.Core.DealerBox Responsible for order request resolution.
{
exception EmptyPositionIsNotAllowedException %Empty position is not allowed% %Empty position is not allowed%;
exception UndefinedPositionTypeException %Undefined trade operation type produces undefined command% %Position: [positionId] of account: [accountId] has invalid tradeOperationId: [tradeOperationId]%;
exception PositionHasInvalidDataException %The data on position is invalid% %The data on [positionId] of [accountId] has invalid parameters%;
ctor (IEngineManager engineManager, IDealerNegotiationManager dealerNegotiationManager, ITransactionManager transactionManager);
```

```
    method Resolve (PositionBase position):void + throw
PositionHasInvalidDataException + throw EmptyPositionIsNotAllowedException + throw
UndefinedPositionTypeException;
    }
    !eresource
```

Listing 4 – Description of task for creation resource

The result of the transformation will be a lot of descriptions of the form domain-oriented description at level 2, which in turn will be converted into the desired shape by appropriate means. As a result programmer receives three exceptions class file, the file with interface, file with class and file with test of this class.

Classes of varying degrees of fullness: exceptions and interface - 100%, the class includes methods of contract with code generation exceptions (implementation of alternative stories), the constructor with initialization logic, the logic of the test is also generated in part offering just a "stub" for direct and alternative scenarios of behavior.

Conclusion. The proposed approach greatly simplifies the reuse of assets, which leads to an increase in the quality of development, focusing programmer’s effort on the implementation of business functions, excluding the details of coding and increasing productivity. Patterns at different levels allow to generate high-grade functional components. The simplicity of template’s description allow to fill decision’s base easily, from the fullness of which depends on the fullness of generating components. The proposed approach is most effective in the case of a preliminary detailed design of components, fullness database solutions, templates, and gives the maximum effect in the implementation of model projects with a low level of research of new technologies.

LITERATURE

1. Visual Paradigm http://www.visual-paradigm.com/.
2. Enterprise Architect http://www.sparxsystems.com.au/.
3. UML http://book.uml3.ru/sec_1_1.
4. CruiseControl http://cruisecontrol.sourceforge.net//
5. SCons http://www.scons.org/.
6. Configuration CruiseControl and SCons http://www.ibm.com/developerworks/ru/library/au-nextgeneration_build/#resources.

7. Mary Beth Chrissis. CMMI® for Development Guidelines for Process Integration and Product Improvement, Addison-Wesley Professional; 3 edition (March 20, 2011). – 688 p.

8. Yii http://yiiframework.ru/doc/guide/ru/quickstart.what-is-yii.

9. FitNesse http://fitnesse.org/.

10. AV Podkopaev Means of describing the code generators for domain-oriented solutions in metaCASEsredstve QReal. St. Petersburg. 2012. - 19 p.Runtime-генератор https://msdn.microsoft.com/ru-ru/library/650ax5cx(v=vs.110).aspx.

11. SOLID http://x-twig.ru/blog/single-responsibility-principle/.