UDK 004.05

A.A. Litvinov, V.V. Gerasimov, D.S. Kovalchuk, V.V. Krokhin

# ON BASIC PRINCIPLES OF MINIMUM VALUABLE INFORMATION SYSTEM DEVELOPMENT AND PREPARATION OF PROFESSIONAL SOFTWARE DEVELOPERS

*Annotation. Basic principles of minimum valuable information system development are summarized and formalized: the system should be developed according to principles of service-oriented architecture; it should be flexible and testable; increasing the flexibility of the system causes the improving of its testing infrastructure; the process of evolutionary development considers stubbing the lower layers components. An example of flexible and maintainable system developed according to the provided principles in short terms is also provided.*

*Keywords: robust system development, software development principles, WCF, WPF, ADO.NET, TDD, MVVM, MS SQL Server.*

**Problem statement.** The era of Information Technology dictates new rules for small business companies, causing the continuous improvement of the software information systems: from conventional Bookkeeping applications to Document-driven and Data-driven decision support systems. Private schools are not exceptions. COTS (Commercial off-the-shelf) products cannot be implemented directly because of domain and business specific, and the only solution is the development of the original information system.

As usual, the set of functional requirements provided by the customer is not complete and refactoring and improvement of the existed units in accordance with the requirements changes could be considered as a business-as-usual situation. For example, the primary set of requirements provided by the private school is as follows. The system should simplify the monitoring of the work done by teachers, visits to students, payments for tuition, teachers' salaries according to skill level and workload. The system should also provide an ability to make and use the schedule of training; and implement a multi-level personal data access control. Usually, when developers ask customer about further improvements such as web-applications and mobile device clients support, students' remote access etc., customer will reply ‘no’.

© Litvinov A.A., Gerasimov V.V., Kovalchuk D.S., Krokhin V.V., 2019

But in the reality, the main part of the lifecycle (especially for modern information systems) is devoted to its evolution and adoption for new requirements after its deployment. Thus, developers face the problem of building highly flexible information system able to be improved and adopted for the business needs without extraordinary expenses.

**Background.** The starting point of the system evolution is MVP (minimum valuable product). According to Ries [1], the MVP is the *"version of a new product which allows a team to collect the maximum amount of validated learning about customers with the least effort."* The MVP should have the core features which allow it to be deployed to some real customers. The purpose of the MVP is getting initial feedback from the customers minimizing the risks of further development. The next steps are learning and measuring customer requirements, defining and prioritizing new objectives. Then the system should be adopted to the customer needs as soon as it possible. How to make the system adaptable to the dynamics of changes is the central question of software development. Most of the patterns, approaches and principles [2-4] are developed to achieve this goal. It is notable that the system should have the potential of flexibility from the start.

**The purpose of the study** is to answer the following questions: What is the structure of minimum valuable information system with the properties stated above, i.e. which layers and units are required to make the system work and ready for further improvements? What principles and methods should be used by the small group of developers to develop such information system? Pure approaches such as TDD, BDD, use-case driven development etc. cannot be used because of the terms and cost restrictions connected to such projects.

In the other hand, during the learning process we encounter the problems of preparation of professionals able to develop such modern systems. Thus, this work can be regarded as an attempt to answer the questions.

**Main part**. Taking the evolutionary development as the foundation, we need to identify the initial point of the evolution, i.e. the skeleton of the system: layers, patterns, development considerations etc. If we look towards the system improvement, we should admit that the only choice is service-oriented architecture. Thus, we can define the first rule of minimum valuable system, it asserts that *each modern application intended to be used collectively should have a potential of its use through remote access using Web protocols*. In addition, using Web-service technologies system increases its openness, i.e. it makes the system opened to other systems as a service and, reversely, allowing to use other systems as services. Taking this assertion as an

axiom we can define three basic layers of the system: user applications, service, server application (business logic). Thus, generally, in terms of set theory, the system can be defined as a system consisted of a set of clients $\boldsymbol{C}$, a set of external services $\boldsymbol{S_t}$ a set of system services $\boldsymbol{S_e}$, and a set of business logic server components $\boldsymbol{B}$ (Fig. 1):

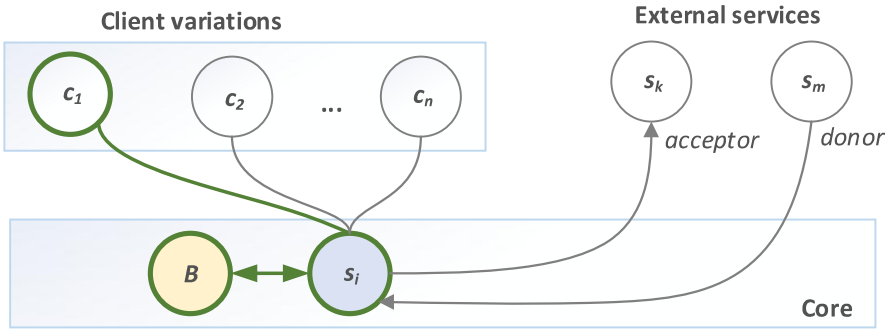


Figure 1 - Structure of generic modern information system

$$\mathfrak{C} = \langle C, \mathbf{S}_t, \mathbf{S}_e, B \rangle \quad (1)$$

It is notable that the minimum valuable version of the system can be described by (2): $s_i$ a definite service without external services utilization, only one variant of user application is realized and business logic represented in its minimum valuable version. The transformation (3) defines the existence of a procedure (set of activities) able to transform the minimal version to its advanced form according to dynamics of changes, i.e. minimal version should have the potential of growth. Generally, we can think about the number of such procedures and the selection of the best one.

$$\mathfrak{C}_{min} = \langle c_l, \mathbf{s}_t, \emptyset, B_{\min \square} \rangle, c_l \in C \ , \mathbf{s}_t \in S \quad (2)$$

$$\varsigma_k : \mathfrak{C}_{min} \to \mathfrak{C}, \varsigma_k \in \boldsymbol{\Sigma} \quad (3)$$

The second rule is connected to the business layer that is the core of the system. While the services are responsible for remote access and primary operations, business logic can be regarded as the core responsible for functionality. It can be divided into three main parts: business simulation, data access layer, database. And the rule states that *business logic must be prepared for change, the system must be ready to change database provider*. Formally, the assertion can be represented by the formulas (4) and (5), where R means business logic and Δ denotes the set of "data access components — storage" relations.

$$B = \langle R, \boldsymbol{\Delta} \rangle, \quad (4)$$

$$\boldsymbol{\Delta} =_{def} \{(a, d) | a \in A, d \in D\}, \quad (5)$$

Minimal version of the business logic layer can be represented by (6).

$$B_{min} =_{def} \langle R, \langle a, d \rangle \rangle, \langle a, d \rangle \in \Delta \quad , \tag{6}$$

The transformation (7) of $B_{min}$ to $B$ means the existence of an effective procedure responsible for the evolution. The effectiveness of the procedure depends on time, efforts and cost spent to achieve the result and can be estimated

$$\beta_j : B_{min} \rightarrow B, \qquad \beta_j \in \mathbf{B} \tag{7}$$

It is obvious that the time and efforts spent on modification the layer and/or its reconfiguration (e.g. changing database provider) depends on testing environment and test coverage of the code involved in the modification. Thus, the third rule states that the *increasing the flexibility of the system causes the improving of its testing infrastructure*.

Next, to make the system and its components testable and maintainable *we need to follow interface-oriented programming paradigm and an additional set of rules represented as SOLID principles*. Simply speaking, the system is represented a set of modules connected via interfaces. The dependencies of a module (classes the module depends on) are injected via constructor on the phase of its instantiation.

The next question is how to solve the tasks of modification the multi-layered system rapidly. The goal is to involve end-user in checking the functionality (use-case, user story) being developed as earlier as it possible. It implies the need to start with UI construction *stubbing the lower layers* (Fig. 2.).

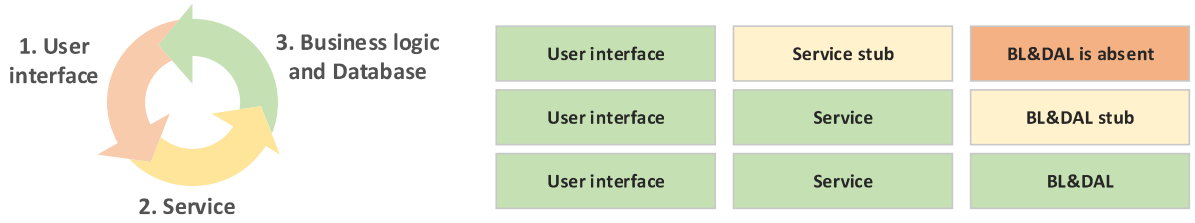


Figure 2 - Basics phases of use-case realization

In such a way to start user interface development we need only to have a stub of the service client and realization of service and business logic layers becomes unnecessary. It significantly simplifies the process. When the UI is realized and end-user commits that it met the requirements, the next step is to realize the service, stubbing the dependencies, i.e. business logic classes it depends on. Thus, there are three basic steps needed to realize the function.

This process allows us to make the system gradually through the evolutionary approach avoiding rushing and misunderstandings. When the work is split among several developers, stubbing also allows them to work independently. Of course, stubbing considers interface-based organization of the system, using de-

pendency injection, because the stub is a variant of realization of an interface injected into the class being developed.

**Experimental part**. As an example, we provide an information system to support the educational process of a private school developed by master's degree student D. S. Kovalchuk. The system was realized in short terms (2 months) by only one developer using the knowledge and experience gained during the first semester of the master course.

Despite the initial requirements provided by the customers D. S. Kovalchuk decided to develop the flexible system based on above principles. Using the principles doesn't mean to give them equal priority. System composed of the four basic layers: UI, service, business logic layer and data access layer. Most of the classes within the layers made flexible based on interface-based programming and SOLID principles. Especially, the attention was focused on connection classes with dependencies. In the result, it increased the level of testability and maintainability of the modules. Estimating the layers against quality attributes such as flexibility and testability, we have the system with a high potential of improvement, but most of modules have insufficient test coverage level (Fig. 3.). High level of flexibility means that the layer can be reconfigured or changed quickly without significant efforts, medium means that not all connections used via interfaces and therefore it is possible that the requirements changes causing the system could not be handled easily. The low level of test coverage of UI and service layers dictated by the short development terms and the specific of the domain, but as we can see the potential testability level is medium, i.e. unit, integration and system tests can be added in accordance with the needs.

| | Flexibility | Testability | Tests coverage |
|---|---|---|---|
| UI | High | Medium | Low |
| Service | Medium | Medium | Low |
| BLL | Medium | Medium | Medium |
| DAL | High | High | High |

Figure 3 - Estimation of the layers against the most important quality attributes

Below we provide detailed description of the project, i.e. its structure, patterns and technologies used.

Communication between the client and the server is carried out using Window Communication Foundation (WCF) technology, which works through web services. Based on this, the system architecture can also be considered as service-

oriented [6]. Thus, in the future, you can increase the functionality of the system, connecting new services.

The designed system operates within the local area network of the enterprise (for example, a Wi-Fi network). Now the network is closed for public access; but it could easily be open for Internet network clients.

As it was mentioned above the connection with the database is carried out through an intermediate layer between the business logic of the server and the database itself — the Data access layer (DAL). Since server modules do not connect rigidly, but through interfaces, you can easily replace a module with any other module that satisfies the contract of a particular interface. Thus, initially using MS SQL Server [8] as a database provider, if necessary, it will be possible to switch to the document-oriented MongoDB system [7]. The implementation of work with this technology, namely the connection to the database, a set of various requests, calls to stored procedures, connection termination — is placed in the previously mentioned layer DAL.

The system has a multi-level personal data access control. This ensures the separation of user access to data according to their roles. For example, when logging in to the user, only those controls that allow him to perform the allowed set of actions will be active (Fig. 4). This system of user roles is implemented by storing in the database the relevant information about the type of user role, as well as using appropriate business logic that processes the role before providing opportunities to work with the system.

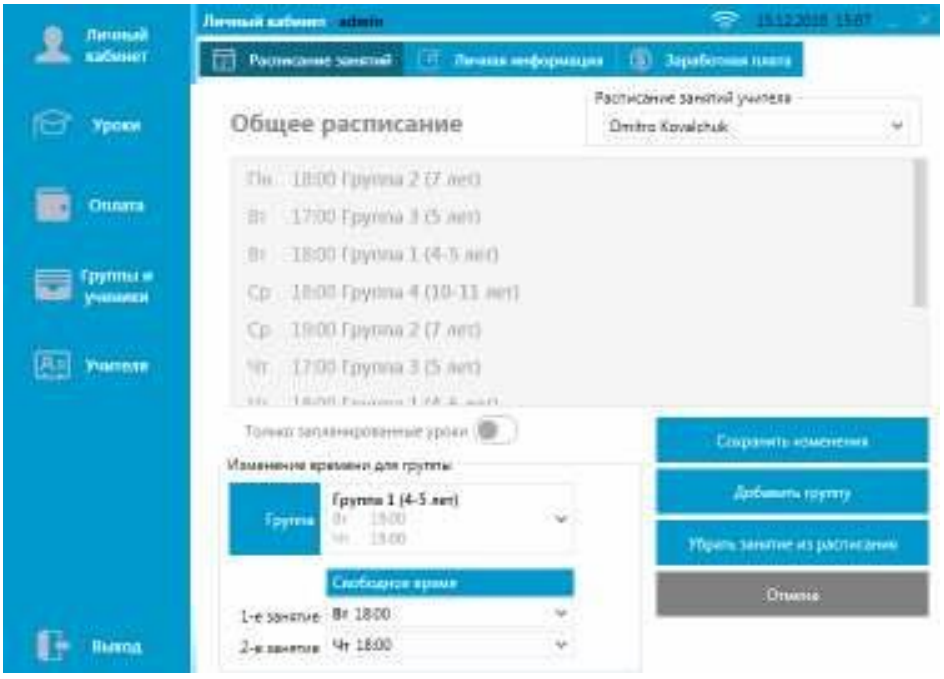


Figure 4 - The main window of the application

The client application was developed using the Windows Presentation Foundation (WPF) technology [9], which provides a friendly and customizable user interface for the application. A Model-View-View Model (MVVM) design pattern [10] was also applied, in which the business logic, the workspace interface itself, and the binding of controls to the processing logic are divided into three components.

At each level of the system, the principle of modularity is applied, that is, it is possible to replace, if necessary, one module or another with another, implementing the necessary contract. That the client, that any module on the server, including the database itself, can be replaced with a new one. This suggests a sufficient flexibility of the system and solves the main problem of the maintenance of the developed product today.

Service-oriented system architecture involves working with services, which in turn are autonomous business logic modules. This allows you to scale the system, expanding the range of its business functions in the future. Thus, it will be possible to "connect" various additional modules, such as corporate chat, work with files on the cloud etc.

**Results.** Basic principles of modern information system development are summarized and formalized. In short, required options are as follows: the system should be developed according to principles of service-oriented architecture; it should be flexible and testable, that means the following the principles of interface-based development and SOLID; increasing the flexibility of the system causes the improving of its testing infrastructure; the process of evolutionary development considers stubbing the lower layers components. These principles were offered in the discipline of robust software systems development (first semester of the master course) and students received such training are able to develop flexible systems in short terms. An example of developed system is also provided. The system is ready to changes, can be easily modified and reconfigured responding to customer needs.

**REFERENCES**

1. Eric Ries. The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses. 336 p.

2. Beck K. Test-Driven Development By Example. –Addison-Wesley.–2002.-240 p.

3. Beck K. Extreme Programming Explained: Embrace Change: 2nd edition. - Addison-Wesley – 2004. — 224 p.

4. Eric Evans. Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley Longman, Reading, MA, First. – 2003.

5. Литвинов А. А., Карпенко Н. В. Тестирование информационных систем: модульное, интеграционное, системное: учебное пособие. / А. А. Литвинов, Н. В. Карпенко — Д.: Лира, 2016. 283 с.

6. Литвинов О. А., Хандецький В. С. Розподілена обробка інформації : [моногр.] / О. А. Литвинов, В. С. Хандецький — Д.: ТОВ «Баланс-Клуб», 2013. 314 с.

7. Open Source Document Database | MongoDB [Electronic resource]. Access Mode: https://www.mongodb.com

8. ADO.NET | Microsoft Docs [Electronic resource]. Access mode: https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/

9. Windows Presentation Foundation (WPF) | Microsoft Docs [Electronic resource]. Access mode: https://docs.microsoft.com/en-us/dotnet/framework/wpf/

10. Model-View-ViewModel - Wikipedia [Electronic resource]. Access mode: https://en.wikipedia.org/wiki/Model-View-ViewModel

**REFERENCES**

1. Eric Ries. The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses. 336 p.

2. Beck K. Test-Driven Development By Example. –Addison-Wesley.–2002.-240 p.

3. Beck K. Extreme Programming Explained: Embrace Change: 2nd edition. - Addison-Wesley – 2004. — 224 p.

4. Eric Evans. Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley Longman, Reading, MA, First. – 2003.

5. Litvinov A. A., Karpenko N. V. Testirovanie informatsionnyih sistem: modulnoe, integratsionnoe, sistemnoe: uchebnoe posobie. / A. A. Litvinov, N. V. Karpenko — D.: Lira, 2016. 283 s.

6. Lytvynov O. A., Khandetskyi V. S. Rozpodilena obrobka informatsii : [monohr.] / O. A. Lytvynov, V. S. Khandetskyi — D.: TOV «Balans-Klub», 2013. 314 s.

7. Open Source Document Database | MongoDB [Electronic resource]. Access Mode: https://www.mongodb.com

8. ADO.NET | Microsoft Docs [Electronic resource]. Access mode: https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/

9. Windows Presentation Foundation (WPF) | Microsoft Docs [Electronic resource]. Access mode: https://docs.microsoft.com/en-us/dotnet/framework/wpf/

10. Model-View-ViewModel - Wikipedia [Electronic resource]. Access mode: https://en.wikipedia.org/wiki/Model-View-ViewModel