Elements of the Modern Application Software Development

O. Moravcik, D. Petrik, T. Skripcak, and P. Schreiber, *Member, IACSIT*

***Abstract*—This article is aimed on the software development process of modern applications. The first part of article starts with the general classification of information system based on user interaction characteristic. After that insight into methodologies, methods, design patterns and tools which are part of modern software development life cycle is presented. Second part is devoted to implementation details of presented modern trends in real world application. Finally selected drawbacks with proposal of theirs solutions are presented. The main goal of this article it to provide overview of current modern trends in software development and point out problems which could be uncovered during adaptation phase of these disciplines.**

***Index Terms*—Composite application, ORM.**

I. Introduction

The domain of information technology belongs to one of the most rapidly developing areas on the world. Nowadays, software companies, which are intended to be successful in the software development market, have to keep their knowledge bases up to date. The task of picking up right methodologies, techniques and tools is critical when we are talking about delivering high-quality and maintainable software products, and still keep time and money costs in reasonable limits. Current modern trends in the development of software applications could help companies in their business but have to be used correctly and the fact that some of them could have negative impact on attributes of resulting software (e.g. performance).

A. Application Posture

The term Application Posture [1] was introduced by Alan Cooper and it basically refers to the way how end users interact with software application. This characteristic is really important mainly because it indicate how important the software is for its users. According to Alan Cooper and Robert Reimann there is following classification of software systems [1]:

1) Sovereign–An application that takes the user's full attention, such as Outlook or Word.
2) Transient–Application in the periphery of the user's attention, calling the user for short moments, such as (for most folks) a calculator.
3) Daemonic–Alerting systems.
4) Parasitic–Support interaction mode for both sovereign and transient applications, such as chat.

From the business perspective, sovereign information systems are the most interesting field for the development process. These applications are planned to be used by many users in a long term period. That is why they have to be designed not only to work well for now, but also to be maintainable in the future.

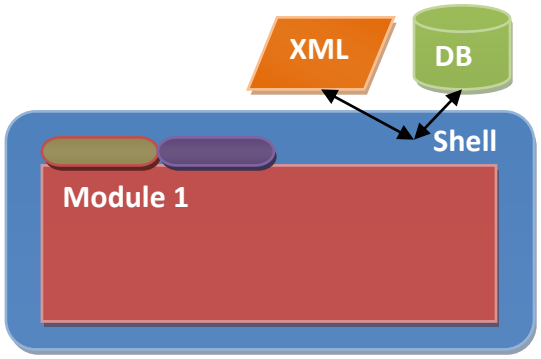

Fig. 1. A basic schema for composite application communication with two data sources.

Designing and building applications in a monolithic style can lead to an application that is very difficult and inefficient to maintain. On the other hand there is another class of system developed according to the composite approach. Composite application is created from group of loosely coupled, semi-independent modules which are easily integrated to coherent solution called *"shell"* [2]. Graphical mock of such an application is figured in the Fig. 1.

B. Modern Methodologies in Software Development

Small and midsized software development companies are often fighting with the need of having high quality methodology in the backend of software development life cycle and the possibility of being agile enough to quickly react on changing requirements from users plus reduce the time needed for iteration cycle in order to produce prototypes of system and provide customers an opportunity to get an insight of the resulting application.

Ideal solution of this problem is to simply compose basis trends, disciplines, methods and tool in way which will be suitable for current software project. Below is the overview of most followed ides for software design:

1) Model-driven architecture (MDA)

Most of modern information systems are developed

Manuscript received July 25, 2012; revised September 25, 2012.

O. Moravcik and P. Schreiber are with the Institute of Applied Informatics, Automation and Mathematics, Slovak University of Technology, Trnava SK 917 24, Slovakia (e-mail: oliver.moravcik@stuba.sk, peter.schreiber@stuba.sk).

D. Petrik is with the MMS Softec Ltd., Trnava, SK 917 01, Slovakia (e-mail: petrik@mms-softec.sk).

T. Skripcak is with the Institute of Applied Informatics, Automation and Mathematics, Slovak University of Technology, Trnava, SK 91724, Slovakia and Department of Information Technology, Helmholtz-Zentrum Dresden-Rossendorf, Dresden, DE 01328, Germany (e-mail: tomas.skripcak@stuba.sk, t.skripcak@hzdr.de).

according to object oriented paradigm. MDA was initially introduced by Object Management Group (OMG) and provide an approach for capturing system-specification via usage of formal models. In MDA, platform-independent models (PMIs) are initially expressed in a platform-independent modeling language, us as Unified Modeling Language (UML). The platform-independent model is subsequently translated to a platform-specific model (PSM) by mapping the PIM to some implementation language or platform (e.g. C#) using formal rules [3].

2) Agile software development

The idea of agility was firstly used by Kent Beck and transformed into methodology called Extreme Programming (XP). This methodology is described like easy, effective, low risk, flexible, predictable, scientific and funny way of software development [4]. Core of agile software development is to use of light, but sufficient rules of project behavior and the use of human and communication oriented rules [5]. In the world of agile development everything is focused on processes in order to deliver good product, that is why, usage of method or process is always depended on current project needs.

3) X-Driven design/development (XDD)

The effort of developing better applications is not only problem of usage of newest technologies. Often in order to develop a good product, insight into stakeholders domain problem is needed. Eric Evans summarized some of well known facts about domain modeling in object oriented world in his book Domain-Driven Design [6]. Independent from DDD, Richard Pawson introduced an idea of Naked Objects in his dissertation thesis [7]. It basically tray to reduce development of software application to creation of complex domain model with metadata information, which is used to generate all other modules including User Interface (UI) in runtime. Systems developed on top of Naked Objects paradigm are presented in special kind of UI known as Object-Oriented User Interface (OOUI) [7]. As a respond to the need of testing the Domain Model early in the software development life cycle, together with progress of dynamically typed languages, Test-Driven Development was introduced mostly build on top of developers Unit Testing. According to [8] a unit test is described as a piece of code (usually a method) that invokes another piece of code and checks the correctness of some assumptions afterward. If the assumptions turn out to be wrong, the unit test has failed. A unit is method or function. There is also one modification of TDD which had spread around the software developer's world called Behavior-Driven Development (BDD) as described in [9]. Main purpose for implementing BDD was complexity and wide range of TDD. BDD is trying to specify good convention in the process of test writing and execution. The other thing is, that according to BDD only functional user requirements are covered by tests. It means that tests practically become functional user requirement specifications which can be read, but also modify easily. These practices go hand in hand with agile methodology for software development.

4) Aspect oriented programming (AOP)

Aspect Oriented Programming is aimed on cross-cutting problems in object oriented applications, which could not be modeled within object oriented paradigm [10]. The example of such problem is a functionality (e.g. logging application messages) which should be applicable on different type of classes (Business Entities, Infrastructure Services, etc.). This type of functionality is encapsulated into routine called Aspect and makes software more reusable and maintainable. Nowadays there are AOP frameworks, for most software programming languages, which form mainstream in application programming. And these frameworks could be taken as enhancement of object oriented paradigm.

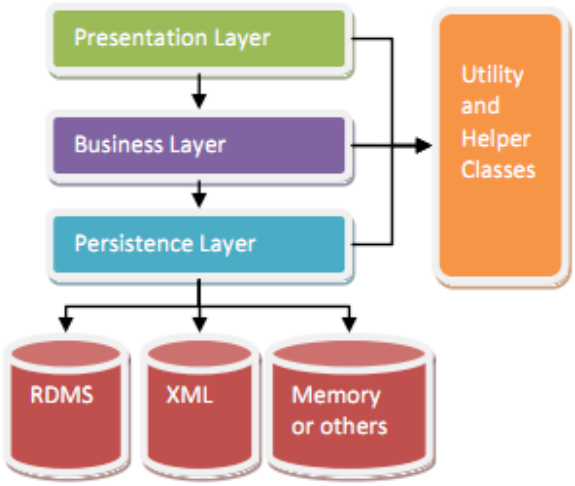

Fig. 2. Architecture of a standard business system with three software layers [11].

5) Object-relational mapping (ORM) as data access strategy

Relational Database Management Systems (RDMS) were and still are standard preferred solutions in the market of corporate information system, mainly because they are mature enough and their wide field of usability is determining factor for many types of software systems. They are building on top relational algebra, which give them solid mathematical background. Modern RDMS were also able to adapt themselves for the needs of current developer's (e.g. there is no problem to work with documents in eXtensible Markup Language XML) RDMS can be used in today's modern object objected applications but developers need to fight with connection of OO and relational worlds. Of course there are other data storage paradigms like Object Oriented Database Managements Systems (OODMS) or Document Oriented Database Management Systems (DODMS) but they are not established yet. It will be interesting to see how this field of informatics will evolve in future, however for now when we are developing OO application with RDMS system as a storage background, it is important to have layer which connect this two different approaches together and that is the case of ORM [11]. The example of layered business application with ORM persistence layer is shown in the Fig. 2.

C. Architectonic Design Patterns

Design patterns are standardized solutions for solving typical not elemental problems in object oriented programming. The special class of design patterns is used for overview of whole architecture for object oriented system. These patterns are called architectonic. Current trends are

mainly based on idea of elimination dependencies between various components of software system. One of the most successful architectonic design patterns nowadays is Model-View-Controller (MVC). The idea of MVC is quite old. It was developed at Xerox PARC in 1978/79 by Trygve Reenskaug, but it gains its position in the world of mainstream software developing only a few years ago. Basic idea was that the model will be abstraction of domain model, the view will contain user presentable interface, and controller will coordinate capabilities of several views making it a comprehensive tool [7]. Variations of MVC pattern, like Model-View-Presenter (MVP) and Model-View-View Model (MVVM – in some literature you can also find View Model under the name Presentation Model) are commonly used in the modern software in order to fully utilize underlying framework of software applications. In the Fig. 3, there is an example of how MVC, MVP and MVVM patterns behave.

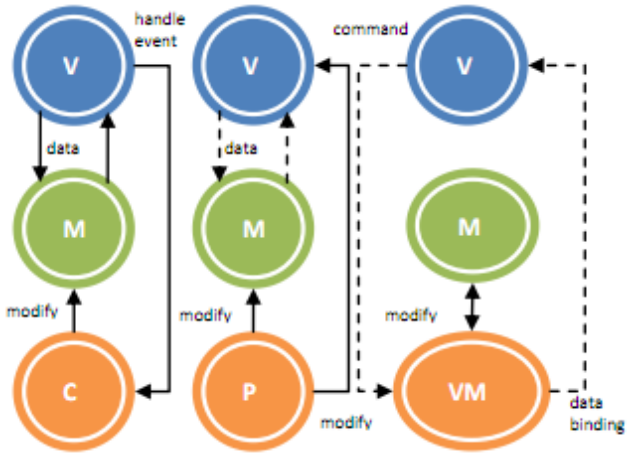

Fig. 3. Behavioral differences between MVC (Model-View-Controller), MVP (Model-View-Presenter) and MVVM (Model-View-View Model) design patterns.

It is worth of saying that in the modern software technique called Dependency Injection (DI) is used in order to remove tightly coupled components and replace them with soft references. For more information about DI please refer to [12].

We would like to point out one more pattern which was introduced by Rinat Abdullin and is called Command and Query Responsibility Segregation (CQRS) [1]. It is still in the process of transforming into its final form, but provides us some interesting ideas. First of all, CQRS differentiate between commands, which purpose is to modify data in storage system, and query producing readable information for users in order to make responsiveness applications. Read operations are much more often needed in comparison with operations for data manipulation. Secondly usage of CQRS entail that domain model and especially domain objects are not presented to end users in their base form. The domain model in CQRS is only used as an abstraction of the domain problem, which contains all business logic for data modifications and also business events. However for the presentation purpose Data Transfer Objects (DTO) are used as a lightweight wrapper to presenting readable data to the user interface [1].

D. Continuous Integration

Term integration is describing an activity for combination software components into system as a complex unit [13]. Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily – leading to multiple integration per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible [14]. Integration is related also with Version Control, which is one of the aspects in Software Configuration Management [15]. Having Version Control system is necessarily when we want to develop high quality software application. There are two main class of Version Control system:

1) *Centralized* – This is built on top of client-server architecture. Server plays role of source code primary repository and each client have to communicate with server in order to perform some action (including commit (check in), check out, view history, revert changes and others).
2) *Decentralized* – Nowadays decentralized version control solutions become more and more popular over centralized. These systems are built on top of peer to peer architecture, where each peer contains repository and changes are distributed from peer to peer as patches. Advantage is that many operations do not need access to network and that is why they have better performance in comparison with centralized version control systems. Disadvantages implies form the nature of distributed solutions, where we do not have one place with most up to date version of developed information system.

II. The Implementation and Elimination of Drawbacks

In the real world, there is always a risk involved when decision to adopt a new technology, methodology or paradigm is made. Each member of team usually has to change the way of thinking about problems and it take some time. Implementation of new tools could also involve some problems with e.g. performance (because we usually create another level of abstraction) and the errors which were not visible during change preparation time will be exposed in the development process.

A. Usage and Limitation of ORM

The main reason for usage of ORM technology was shielding the software developer form any interaction with database level of an application. The developer can be fully focused to domain model, which contains all domain logic and do not have to bother with SQL (Structured Query Language) commands and relational database schema [16]. That is why developer is much closer to the world of the end user and can express his idea in language (called Ubiquitous Language in [6]) understandable for both of them.

There are many ORM frameworks on the market, so preliminary research and tests are necessary before choosing the product which will be used in the software project. For

the purpose of persistence strategy, we have chosen ORM framework described in [17]. In the following list, there are outlined features important for ORM, when it will be used as persistence mechanism for composite system, which is developed according to modern trends [16]:

1) Support for many database systems and persistence storages (at least MS SQL and Oracle).
2) Schema creation and schema update – Ability of creation database schema and update it if necessary from class model of domain object. It enabled us to have one object oriented domain model modeled in CASE (Computer Aides Software Engineering) tool. It meets MDA but also Agile way of thinking about software development.
3) Automatic mapping relations between objects (1:1, 1:N, M:N) to the database structure.
4) Support for transactions – Unit of Work design pattern (for more information please refer to [18])
5) Superior query mechanism (ideally strongly typed) – Query should not be typed as string. Enables compiler checking of queries (in our case, we required implementation of LINQ [19] technology).
6) Automatic notification when change in attribute value occurred – It enables rich UI experience in Composite Application.

During the development process we have discovered some limitation and problems which were involved into application of ORM into real life development cycle. They are listed in the following list [16]:

1) Processing large amounts of data – Modification of any object's attributes require the reading of the object in memory. This can lower performance, e.g. when aggregation root domain object with many associated objects or with a more deeper hierarchy (object in tree) is deleted, all object waiting for deletion should be first read from database to the memory and the deletion process afterwards. We solved this issue by analysis of application bottlenecks and defining cascade delete rule in the relational database system.
2) The impossibility to join objects, which do not have relations between them, in the query. The only working solution we found out is the definition of database view. It can be wrapped into the object and used in query afterwards.
3) The problem with querying calculated attributes – In some cases calculated attribute of the domain object is needed as a part of query expression. In order to gain value of calculate attribute whole object have to be read from database, which has significant negative impact on query performance. The solution is to define procedure which will be calculating the value on relational database system but the negative side is that we will have duplicated business logic. That is recommended only in bottlenecks of application.

B. Modern User Interface Composition

Current *"rich"* business applications typically feature multiple screens, rich, flexible user interaction, data visualization and role-determined behavior. The application's expected lifetime is measured in years and that it will change in response to new, unforeseen requirements. This application may start as small and over time evolve into a composite client [16].

We have based our project on top of guidance [2] for building next generation application in WPF (Windows Presentation Foundation) technology provided by Microsoft. It contains set of standards, design patterns and libraries which help solving common problem in composite application development. Probably the most important is the implementation of architectonic design pattern called MVVM. We slightly modify it to meet all our needs:

1) View

XAML (eXtensible Application Markup Language) is used for the purpose of View implementation. We decided to use "in view" constructing of UI and composition technique like templating, styling and data binding with automatic change notification. WPF provide us all necessary foundation. View should contain only functions directly connected with the application UI.

2) View model

View Model class have access to all data and actions which user can perform through UI. It transforms the data in for the purpose of displaying them to the UI. We implement View Models class as simple and reusable as possible. In the context of composite application VM is created with help of DI approach and it automatically run the initialization of View component which is provided to end user to work with immediately.

3) Model

Model is defined and generated from UML (Unified Modeling Language) Class diagrams. It contains all business logic and is extended with check of validation and business rules. It also serves as basis for ORM database generation and manipulation.

Model-View-View Model design pattern is not applicable only for architecture of software system. Some of new software framework successfully uses this approach to obtain higher configurability of its components. As an example we can state WPF framework for building user interface. In the terms of WPF every control or component which will be displayed on the screen is by default look less. When we need to define a graphical shape of the user control, we crate element called control template. Control templates are defined by framework of course, but there is still a possibility to override these settings and modify them to meet our needs. The other purpose of control template is specifying the place where the data will be presented on the control. Most LOB (Line of Business) applications are data driven and binding declarative binding of data to UI and also changes from UI back to data, is another aspect of modern UI composition. In WPF data template can be used to define UI elements in which provided data will be presented. Data can by also shaped with utilization of converters. At last the style is applied on the user control to create delightful graphical feeling. In the context of object oriented model we would say that composition is the feature which is used in modern UI paradigm, where user control is directly composed from other base controls and this process can be done recursively until the final stage is reached.

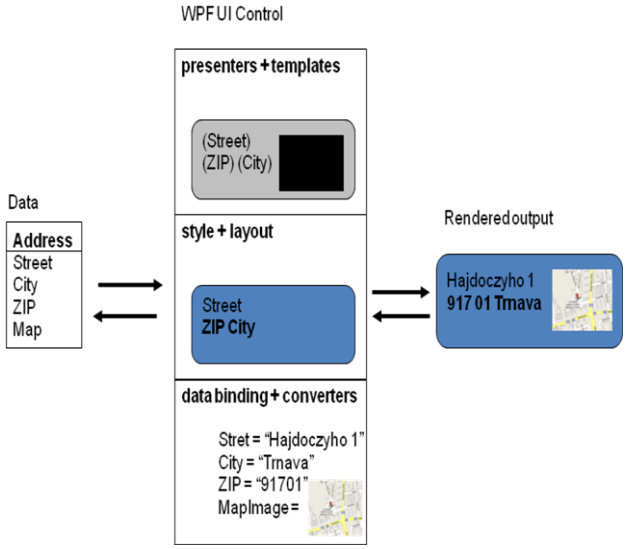

Fig. 4. An example of the visual output composition [19].

Fig. 4 shows how all this could work together in order to provide consistent and responsive user interface. This way of thinking about UI could be applied to different platforms e.g. mobile, web and client. WPF is oriented on rich desktop applications but it is a big step forward. When we are considering web, which is actually much lest interactive (in the context of web applications), have its own user interface description language (HTML – Hypertext Markup Language) and styles written as CSS (Cascading Style Sheet) rules are used for defining graphical visuals for many years, desktop UI development , on the other hand was very limited until present times.

C. Managed Languages and Performace

In object oriented applications written on top of software framework there is a component responsible for memory management called *"garbage collector"* (GC). Its main goal is to delete unused object form memory heap to in order to prevent performance problems. However there are situations when dependency between objects does not allow to GS to dispose objects from memory. We had also discovered this problem, when we were dealing with implementation of MVVM pattern in composite WPF application. We found out that some View-Model components exist in memory more than once, which has negative impact to performance. We had examined this behavior and it seems that it occur only when data object from View Model is presented by user control (technical speaking a *"Data Context"* property of any user control keep reference to the data object), but does not have implemented *"Inotify Property Changed"* interface (.NET interface for propagating change in data immediately to the UI) [16].

According to [20], [21] WPF uses the *"Value Changed"* event, which involves calling the *"Property Descriptor. AddValue Changed"* method on the *"Property Descriptor"* object that corresponds to property. Unfortunately, this action causes that the Common Language Runtime (CLR) also keeps a reference to the *"Property Descriptor"* object in a global table.

The diagnosis of memory leaking seems to be simple especially when a memory profiler is used. This tool can help to find out *"paths to a GC root"*, which can be used to identify a problem object and its references [16]

III. Open Issues

Developing of modern information systems is complex process and many tasks are still very difficult to accomplish. For example it is not well defined standard on executing parallel operations and also asynchronous user interface.

Another issue involve complex testing (involving testing components for other vendors) of an application. Currently an ideal solution candidate for this task is utilization of UI testing tool.

IV. Conclusion

The aim of this article is to provide overview of modern disciplines in applications development process. It outlined some problems which are likely to come along during learning and adopting of these techniques and solutions are offered in the context of real world information system. Modern trends in application development industry have of course positive effect and try to simplify routine developer's task. But we have to make clear that every progress need time so that software companies could adopt and use it correctly.

References

[1] R. Abdullin. (October 2009). Command and query responsibility segregation. [Online]. Available: http://abdullin.com/cqrs
[2] M. Patterns and P. Team (November 2010). Prism 4.0. [Online]. Available: http://www.microsoft.com/en-us/download/details.aspx?id=4922
[3] J. D. Poole, "Model-driven architecture: Vision, standards and emerging technologies," in *ECOOP Workshop on Metamodeling and Adaptive Object Models*, pp. 2-7, 2001.
[4] K. Beck, *Extreme Programming, Extrémní Programování*, Praha: Grada, pp. 158, 2002.
[5] A. Cockburn, *Agile Software Development*, Addison-Wesley, pp. 8-9, 2001.
[6] E. Evans, *Doman-Driven Design: Taskling Complexity in the Heart of Software*, New York: Addison Wesley, pp. 560, 2003.
[7] R. Pawson, "Naked objects," Ph. D. dissertation, University of Dublin, Trinity College, Dublin, 2004.
[8] R. Osheroves, *The Art of Unit Testing*, Manning, pp. 320, 2009.
[9] C. Kaner and J. Bach. (2009). Introduction to BDD. [Online]. Available: http://dannorth.net/introducing-bdd
[10] G. Kiczales, J. Lamping. A. Mendhekar, C. Maeda, C. Lopes, J. M. Loingtier, and J. Irwin, "Aspect-oriented programming," in *European Conference on Object-Oriented Programming ECOOP*, Finland, pp. 25, 1997.
[11] P. H. Kuaté, T. Harris, C. Bauer, and G King, *NHibernate in Action*, Manning, pp. 400, 2009.
[12] M. Seemann, *Dependency Injection in .NET*, Manning, 2009, pp. 375.
[13] S. M. Connell, *Code Complete, Dokonalý kód*, Brno: Computer Press, a.s., pp. 694, 2006.
[14] M. Fowler. (2006). Continuous integration. [Online]. Available: http://www.martinfowler.com/articles/continuousIntegration.html
[15] W. Lewis and G. Veerapillai, *Software Testing and Continuous Quality Improvement*, 2nd ed. Auerbach publications, pp. 534, 2005.
[16] D. Petřík and O. Moravčík, "Modern trends in the development of software applications," in *Proc. International Workshop Innovation Information Technologies – Theory and Practice*, Dresden, pp. 3-5, 2010.
[17] Dev Express. (2006). Express persistence objects. [Online]. Available: http://www.devexpress.com/Products/NET/ORM
[18] M. Fowler. (2002). Unit of work – design pattern. [Online]. Available: http://martinfowler.com/eaaCatalog/unitOfWork.html
[19] F. Marguerie, S. Eichert, and J. Wooley, *LINQ in Action*, Manning, pp. 576, 2008.

[20] J. Goldberg. (2005). WPF performance and. NET framework client profile. [Online]. Available: http://blogs.msdn.com/b/jgoldb/archive/2008/02/04/finding-memory-leaks-in-wpf-based-applications.aspx
[21] Microsoft. (2005). Memory leak in WPF. [Online]. Available: http://support.microsoft.com/kb/938416/sk

Oliver Moravcik was born in Male Levare, Slovakia, 1952. He received his diploma degree in the field of automation at University of Technology Ilmenau/Germany and Ph.D. degree in the field of computer processing also at University of Technology Ilmenau/Germany. He became an associated (1990) and full professor (1998) at Slovak University of Technology in Bratislava/Slovakia. He was visiting professor at Technical University in Koethen/Germany (1986-1988) and at University of Applied Sciences Darmstadt/Germany (1990-1992) as Konrad Zuse fellowship winner. He is focused in the field of software technology and software development. He published more than 100 articles, books and reports in international and national journals. He leaded about 25 national and international projects. Professor Moravcik is since 2006 the dean of the Faculty of Materials Science and Technology in Trnava of the Slovak University of Technology in Bratislava.

Daniel Petrik was born in Ilava/Slovakia in 1968. He received his diploma degree in the field of Manufacturing Systems and Robotics from Slovak University of Technology in Bratislava in 1991. After finishing his studies he worked at the Slovak University of Technology in the field of process automation and information (1991-1996). In 1994 he started his part time job in MMS Softec Ltd (Trnava/ Slovakia) as a junior software developer. In November 1996 he left the Slovak University of Technology and became a software architect in MMS Softec Ltd (Trnava/Slovakia). In MMS Softec Ltd. he designed the architecture of the following information systems: easy, e-VEGA, Proman NG®, participated on the design of the system Proman W®. He is publishing papers in national and international conferences and journals. MSc Daniel Petrik's research area of interest is software systems design, testing and its automation.

Tomas Skripcak was born in Trnava, Slovakia, in 1986. He received his Diploma degree in the field of applied informatics and automation in industry, from Slovak University of Technology in Bratislava in 2010. During studies (2005-2011), he worked as a software developer in MMS Softec Ltd. (Trnava-Slovakia), where he was responsible for design, development, documentation and testing of information systems based on .NET technology. In certain time, he is a PhD student at Slovak University of Technology in the field of process automation and information. Since February 2011, he is situated in Germany at Helmholtz-Zentrum Dresden-Rossendorf. He is publishing papers in national and international conferences and journals. His research area of interest includes virtual reality system design, application of machine learning and novel approaches of human-computer interaction.

Peter Schreiber was born in Bratislava, Slovakia, in 1960. He received his diploma degree in the field of Control Theory from the Slovak University of Technology in 1984 and a PhD. degree in the field of Automation from the same university in 1992. He became an associated professor of the Slovak University of Technology for Applied Informatics and Automation in the year 2000. He has worked as assistant lecturer and lecturer in the Slovak University of Technology. In the years 1995 – 1996 he was a lecturer in Koethen in Germany. The spheres of his interest are software systems development and intelligent control methods. He published more than 100 articles in international journals and in the proceedings of international conferences. He participated or leaded the solution of more than 10 grant projects in the areas of software development and control systems.