Efficient Development and Maintenance of Enterprise Information Systems in Multicultural Developer Environment

Milovanović, M., Miloš; Milutinović, M., Veljko

Abstract— Development of enterprise systems has following characteristics: deadlines are always too short, there is no final specification, requirements change in the development phase and nobody knows the whole business process exactly (at least those who know the business process don't know how to specify software which would support that process). On the other side, developers in big international corporations come from different countries; have different education and culture background, as well as their way of software development. In such environment it is extremely important that each part of the system is very flexible, generic and parameterized so runtime changes can be handled instantly. This paper gives a view on serious problems in the development process and the methodology for effective problem solving. There are several frameworks that help developers in building the enterprise systems, but if a developer is not aware of these problems, framework cannot prevent him or her from making the similar mistakes. Methodology presented in this paper is completely independent from the technology.

Index Terms—Information systems, development process, MDD, OOP, OOD, generic applications

1. INTRODUCTION

DEVELOPMENT of enterprise information systems is a challenging job with variety of problems which can appear anytime. For example, web portals, which are very popular nowadays, can be extremely difficult to develop because they should be often integrated with some old legacy software system. Usually nobody knows how that legacy software works exactly, which makes a lot of trouble to developers.

The biggest problem in the development of web portals is that they are, by default, huge; so, at the first sight, they can be developed very fast and a lot of parts can be developed in parallel and later on easily integrated. The following sentence is very popular in the management population, but each developer should be alarmed when he or she hears something like it. .

"Development of that software is **EASY**; it is very **SIMILAR** to something we did before; we should **ONLY** add some **SMALL** changes."

Magic four words and its usage. The sentence given above and its variations are frequently used by the managers, but developers should be suspicious when they hear them. Whenever you hear a word EASY, SIMILAR, ONLY or SMALL, it is better you look closer into your software specification (if it exists).

The second biggest problem is that nobody knows all details about the business which should be supported with the web portal. This can cause many changes in the system requirements in the development phase. This usually happens in the presentation layer, when some manager or tester sees that something is displayed in the wrong place, or some functionality should be disabled in some cases Bearing this in mind, we can conclude the following:

"Everything that is displayed on the screen should be parameterized because there is a high probability that it can be changed."

A simple example is when application is developed in English language and later it is required that application has Multilanguage support.

The third very important problem is a communication problem between users – managers – developers. This problem appears because in big multinational companies there are people from different countries, with different education and culture. This can cause significant problems in communication. Unified modeling language (UML) tries to resolve these problems but still it does not solve all of them.

In the environment like this, a developer is forced to write highly parameterized code, which can be easily customized during the whole software lifecycle. This paper presents the methodology for effective software development,, which is highly flexible, generic, reusable and above all, it is extremely easy to build the new software parts on top of the old legacy software systems. This methodology makes it easy to build high quality software on top of software and databases with significant design problems.

2. Significant Problems In Development Of Enterprise Systems

This chapter presents characteristics of the development process, which can cause a lot of difficulties to the developers. Each developer should be aware of these problems and be prepared to solve them if they appear.

2.1 Rapid application development

As it was mentioned before, managers often say that something is easy, that it can be done fast and if possible, finished within an unreasonable timeframe (very often, when asked for a deadline, managers reply: "The deadline is yesterday!")... This is very common requirement and can cause many problems. Even if something can be done "quick and dirty" developers should avoid this development style because they can regret it in later phases. Figure 2 shows the progress of the software project in the typical scenario and in the scenario using our methodology.

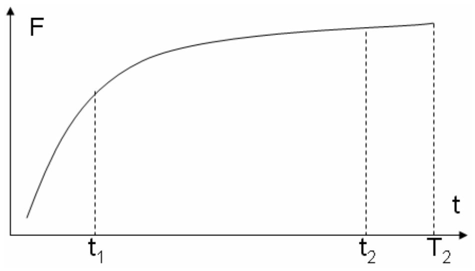


Fig. 2a. Progress of the software project in case of "Quick and Dirty" development style. Each software process, roughly speaking, has tree phases: setup phase, central development phase and final tuning phase. In case of a "quick and dirty" development style we can notice that in the first phase everything looks excellent because lot of functionality is developed fast. But in later phases, it becomes very difficult and time consuming to make even a small change.

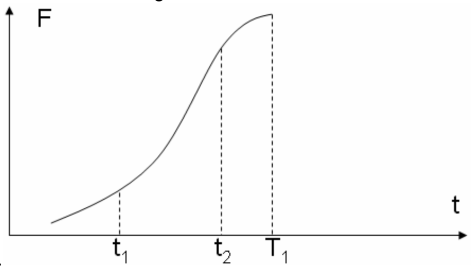


Fig. 2b. Progress of the software project using our methodology. In our methodology, a development phase can be much shorter. The only problem is that in the early phases of development there is no significant progress from the manager's point of view so they become suspicious and require quick and dirty development style. Phase 1 (before t_1) is used for the development of the framework and modeling of the software. Phase 2 (between t_1 and t_2) is a phase of the exploitation of framework potentials and most of the functionalities are implemented in that phase. Phase 3 (after t_3) is used for the final tuning and needs to be done manually.

As it is shown in the figure 2.a., it seems that quick and dirty approach is better, because it is faster in the first phase, but later our methodology proves to be much better because everything is implemented clean, and modeled properly so later variety of functionalities can be implemented extremely fast. In case of a "quick and dirty" style, in the later phases not only development cannot progress, also in many cases some code parts need to be rewritten from the scratch, which is very inefficient and time consuming.

2.2 Requirements are changed during development phase

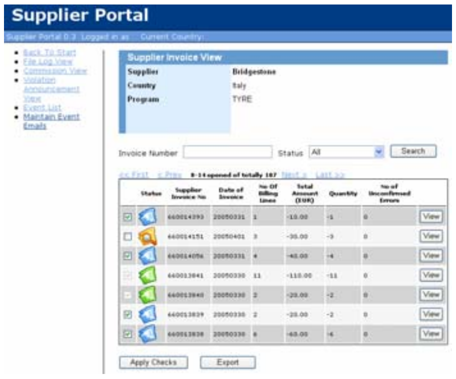
Very frequent scenario is that manager notices that some "small" requirement should be added in the development phase, and even in the test phase. This can happen because managers don't know the business process in detail or don't know much about the capabilities of their developers so they give just rough specification in the beginning and later add more details. Typical example is that if some table is displayed, then some new column should be added or removed, or a number of displayed rows should be changed or some simple functionality should be added.

2.3 Extension of the existing system

Typical requirement is that the existing system should be extended with new functionality. Nowadays all companies want to have web portals for doing eBusiness. This is very tough requirement because existing systems can be very old and their design can be obsolete for a long time, even technology which was used for that old system can be abandoned. So, a developer needs to be extremely careful, needs to make strict interface to the old system and to make communication protocol with the old system highly parameterized, as will be described later

2.4 Fancy and user friendly front end

The last, but the most important thing is that a user interface needs to have a fancy design and to be user friendly. An example of the typical enterprise web portal is presented in the figure 3.



Supplier	Bridgestone
Country	Italy
Program	TYRE

8-14 opened of totally 187

	Status	Supplier Invoice No	Date of Invoice	No Of Billing Lines	Total Amount (EUR)	Quantity	No of Unconfirmed Errors	
✓		440014393	20050331	1	-10.00	-1	0	View
☐		440014151	20050401	3	-30.00	-3	0	View
✓		440014056	20050331	4	-40.00	-4	0	View
		440013841	20050330	11	-110.00	-11	0	View
		440013840	20050330	2	-20.00	-2	0	View
✓		440013839	20050330	2	-20.00	-2	0	View
✓		440013838	20050330	6	-60.00	-6	0	View

Fig. 3. An example of the user interface of the modern enterprise web portals. Variety of data is displayed and a lot of functionalities are offered to the end user.

Modern enterprise systems have plenty of functionalities and a nice design. It is very difficult to develop and maintain both of these characteristics, due to the problems mentioned above. An old legacy software system can hide behind the nice fancy user interface and many changes should be performed in the new application in order to make the whole system working properly. That's why a new application needs to be extremely flexible.

3. Proposed Solution

We propose software architecture presented in the figure 4. Proposed architecture has four major layers: Presentation Layer, Business Logic, Framework, View Layer and Database. Purpose and responsibility of each layer will be presented in this chapter.

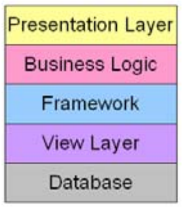


Fig. 4. Software architecture. This software architecture is suitable to meet all requirements for building the enterprise information system and to successfully handle all mentioned problems.

1) *View layer*

View layer is the first layer on top of the Database. This layer is extremely important because it separates the code from the database. This is useful because database is shared and its structure can be changed anytime, so it is very good development style if we make those changes transparent for the code. Also, database can have significant problems in the design and it is much better to fix some problems first and then build the application on top of the clean database. Following transformations of the database structure can be very useful and make the rest of the code much better.

a) Merging structures in the database

It often happens that attributes of one entity are spread over several tables. In that case it is much better to join those tables first, using view, and then build application which sees one entity in one view/table.

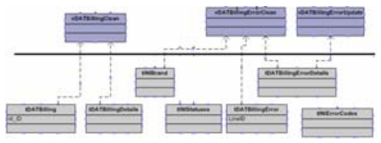


Fig. 5. Joining tables in the database. If one entity is spread over several tables, it is much better to join those tables using view first and then build the application which sees one entity in one view/table.

b) Extraction of key entities

Very common problem is that the database design is not normalized. It means that several entities are joined and stored in one single table. In that case it is much better to use views to extract these entities first, to have clean model and then build application on top of it. Figure 6 presents this situation.

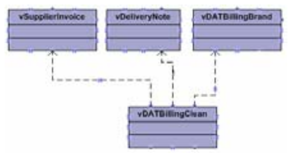


Fig. 6a. Extraction of entities from a single table. In case that database has a design problem in which several entities are joined and stored in the single table, using views can extract those entities.

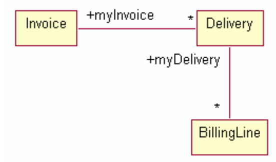


Fig. 6b. UML model achieved using views. After we perform the extraction of entities using views we will have clean model of software, and each entity will be stored in the single table. Code which should be built on top of that database will be much cleaner.

c) Recycling the data

Common case is that in the database we have tables with huge number of columns (we once saw a table with 78 columns!) and we don't need most of them. Also, some column values can be packed or some abbreviations can be stored (legacy data). In those cases it is much better if data are refined first before they go to the application. Views can help us for that. Figure 7. presents view layer on top of table layer in the database.

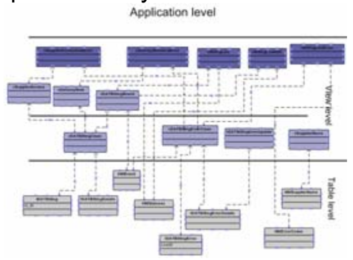
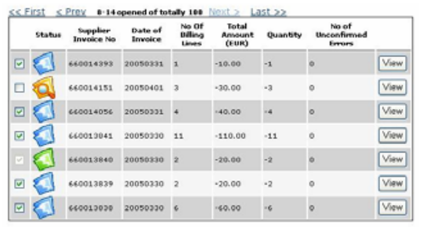


Fig. 7. View layer. Even we have just a few tables in the database, view layer can be big and also contains a sub layers. Each layer will perform one of the transformations mentioned in this section.

2) *Framework*

Our ultimate goal is to develop the application which looks like presented in the figure 8. We can notice there are a lot of elements of the presentation layer and a lot of functionalities. To put them all together we will add one more abstract layer which will speed up the later development.



<< First < Prev 8-14 opened of totally 100 Next > Last >>

	Status	Supplier Invoice No	Date of Invoice	No Of Billing Lines	Total Amount (EUR)	Quantity	No of Unconfirmed Errors	
✓		660014393	20050331	1	-10.00	-1	0	View
☐		660014151	20050401	3	-30.00	-3	0	View
✓		660014056	20050331	4	-40.00	-4	0	View
✓		660013841	20050330	11	-110.00	-11	0	View
✓		660013840	20050330	2	-20.00	-2	0	View
✓		660013839	20050330	2	-20.00	-2	0	View
✓		660013838	20050330	6	-60.00	-6	0	View

Fig. 8. Ultimate application. Our final goal is to design the application which looks like one presented on the picture. Obviously we have in here elements of the presentational layer (data and images) and the business logic (buttons).

Responsibility of this layer is to be a one more abstract layer which will help in the rapid development of business logic and presentation layer. Upper layers will be developed by customizing required look and feel and required functionalities in this framework.

Two major classes in the framework are `TableView` and `Command`. The idea is that class `TableView` should be responsible for drawing of its items and class `Command` should be responsible for the proper behavior. Common case is that for each row presented in the table several functions can be executed. That's why we developed class `TableView` as the container for objects of the class `Command`. In that way a class `TableView` will receive several object of the class `Command` and for each of them will draw a button next to each row. When the button is pressed, a `TableView` object will detect which button is pressed and a proper command will be executed. UML diagram is presented in the figure 9.

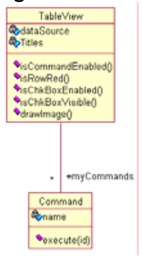


Fig. 10. UML modes of the base classes in framework. Classes `TableView` and `Command` are abstract classes. All classes responsible for the presentation will be derived from the class `TableView` and all classes responsible for the behavior (business logic) will be derived from the class `Command`.

The `TableView` class is responsible for the presentation and class `Command` is responsible for the behavior. UML diagram of the derived classes is presented in the figure 11.

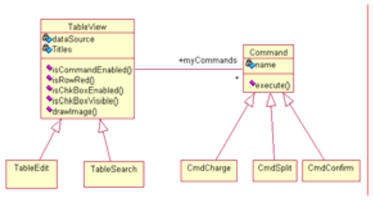


Fig. 11. UML Class diagram. Left side of the diagram is reserved for the classes responsible for the presentation and the right side is reserved for the classes responsible for the behavior. Design of classes `TableView` and `Command` is completely orthogonal. It means that each derived class from the presentation part can be combined with the each `Command` class.

This way we have the strict separation between presentation and the business logic. If we want to add new functionality we just add new class and everything else remains the same. Also, if something in the presentation should be changed, a new class should be derived and everything in the behavior part remains the same.

3) Business logic

Central part of the business logic layer is the class `Command` which implements Design Pattern Command. This abstract class has the abstract method `Command::Execute` which should be redefined in each derived class. Class implements the design pattern Command which allows easy development of functionalities: logging, security protection, unto and all other functionality related to command execution.

4) Presentation Layer

Base class `TableView` is central in the presentation part. That class is responsible for the look and feel of the system. Other classes like `TableEdit` or `TableSearch can be derived` from that class , and they will have a specific look and feel for the specific functionality they support. We noticed in the discussion on the problems that **every human visible item should be parameterized**. This layer and the class `TableView` provides that. It contains methods which control if any element will be presented and how it will be presented. By default, all elements are visible but in the derived classes the way of presentation can be changed.

5) Database Manager

Often requirement is that system should be independent from the specific RDBMS. It means that code should be able to use Orcale, MS Sql Server, SQLite and other databases. To achieve that, another layer needs to be introduced. That layer will hide specific database from the rest of the system.

4. Conclusion

This approach is successfully applied in the development of information system for Ford. Our assumptions about possible problems were correct and proposed architecture solved all of them. We proved that development of enterprise applications using described architecture has many advantages:

- Development is fast. Only the first phase seems to be slow. In other phases, the development is extremely fast and required functionalities are developed also very fast.
- Changes in the requirements during the development phase are handled easily.
- Problems related to the database design are solved efficiently.
- Problems in the communication with other developer can be handled easily because architecture is very flexible with two levels of indirection: View Layer and Framework.

Acknowledgment

We would like to thank especially to Dragan Milićev, Dejan Marjanovic and Nebojša Piroćanac, excellent developers and computer scientists, for teaching me how to build high quality software for enterprize sistems.

References

[1] Milićev, D., *Object Oriented Modeling in Language UML*, Mikro knjiga, Belgrade, 2001.
[2] Lazarević, B., Marjanović, Z., Aničić, N., Babarogić, S., *Database systems*, Faculty of Organizational Sciences, Belgrade, 2003.
[3] Gamma, E., Helm, R., Johnson, R., Vlissides, J., *Design Patterns*, Addison-Wesley, 1995.
[4] Robinson, S., Allen, K., Cornes, O., *Professional programming*, *C#.Net*, Cet, Belgrade, 2005.

Milovanović M. Miloš was born in Kragujevac, Serbia in 1980. He is a PhD student at Barcelona Supercomputing Center, Barcelona, Spain. He graduated at the Faculty of Electrical Engineering (ETF), University of Belgrade in 2004. He was teaching and research assistant at ETF from 2004 till 2006, where he was teaching six courses related to Computer Science and was involved in several international projects. He attended variety of international programming contests and won lot of awards. The most significant awards were silver medal at International Olympiad in Informatics (IOI) in Turkey in 1999 and 5th place in Nokia Code Master Competition among 700 contestants.