УДК 339.341

Buzyka V. V., Chief Technical Officer (Segater LLC, Dnipropetrovsk)

DOMAIN DRIVEN DESIGN AS A BASIS FOR SOFTWARE DEVELOPMENT

This article considers the problem of designing complex software systems using methodologies of domain-driven design. It highlights the challenges faced by developers of large software systems and solutions in the context of the proposed methodology. It also addresses main problems of implementing domain-driven design in manufacturing, cases where this methodology is not applicable or inappropriate.
***Keywords*: domain-driven design, model, universal language, object oriented programming.**

Introduction. Information projects ceased to be a business of one person, who represents a customer of a designer along with a project developer long ago. Nowadays, it is necessary to apply efforts of many people, such as business experts, developers, analysts, and other project participants to develop an IT-project. A contribution of each expert is very important for the development of high-quality and high-demand business product. Frequently many of them do not have enough knowledge in IT-industry. In connection with this, misunderstanding rises among the members of the team creating a project. As a result:

- Task assignments do not reflect needs.
- Completed system does not correspond to task assignments.

Tasks and objectives. When we see the aforesaid problems on the example of many IT-projects and start-ups, which have recently appeared in abundance in the Internet, we would like to find a way to solve the problem of different understanding of various tasks, aims and project potential by its participants.

One of the ways to solve this problem is domain-driven design (DDD). This approach is an evolution in the field of domain modeling and domain design on its based on it. Main conceptual ideas of DDD were generalized and collected by Eric Evans in his book “Domain-Driven Design: Tackling Complexity in the Heart of Software” (2003).

Results. The aim of software product creation is domain automation. As any real object, the domain is mostly many-sided. To create valuable program for users of any field of activity, a developer group shall use knowledge that refer to this field. The volume of the required knowledge

may be rather wide and complicated, especially for big projects. To cope with this volume, DDD offers to apply a Model tool. The Model is a simplification, the interpretation of reality, wherein essential for task solution aspects are extracted from a phenomenon and unnecessary details are ignored. This concept is basic for DDD.

The Model shall be determined by 3 concepts:

1. The Model and software architecture mutually define each other (**Model-Driven Design**).
2. The Model underlies the language of all development group members (**Universal Language**).
3. The Model is distilled knowledge. It represents a coordinated method of structuring of domain knowledge and selection of domain elements that represent the greatest interest.

However, before creating the model, the developers shall acquire this knowledge, since the developer is rarely an expert in the field to be automates. Usually, business-experts are the source of such information. The business experts along with the developers draw information and process it into a convenient form. They sink into data flow and search for the most important stream. Then, they test various methods of data organization in order to find a simple presentation, which would assign a specific meaning to all data. A new system of abstract concepts that considers all required details brings success. This process guarantees strictly formulated knowledge that is the most important in the field and cleared from everything unnecessary. This is one of the most important processes in DDD – the process of **Knowledge processing**. **[1]**

Let's give a small example of knowledge processing. Developers get a task to develop a system of calculation of a employee's salary in the stated month. We find out from business experts how to calculate this metrics "The ratio of the number of days worked to the number of working days in a month multiplied by the ratio of the worker's salary to the number of days worked in the month". On the basis of this we construct the following model:

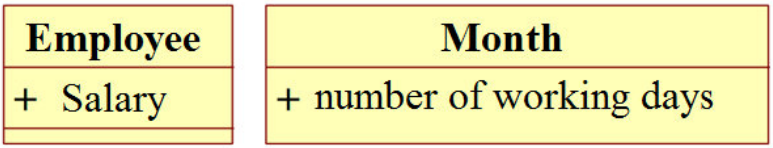

Fig. 1. Initial model of object domain

Thus, we have a question – what is a day worked? The business experts' answer is "The day worked is the day when the employee was present at his

work place, at the same time he has not resigned his job yet". According to the data received we change the model of the object domain in the following way:

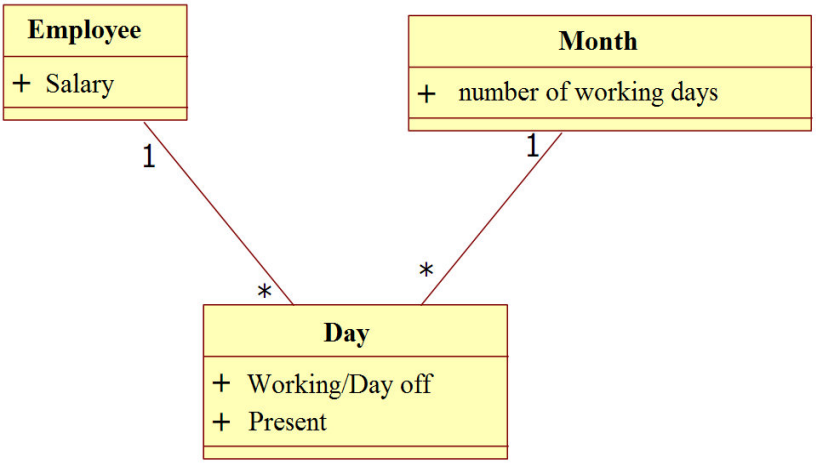

Fig. 2. Domain model (iteration 2)

Having examined the model the domain experts recall that the employee might take a holiday or be on sick leave provided, that such days are calculated at a special rate. Again, the developers change the model:

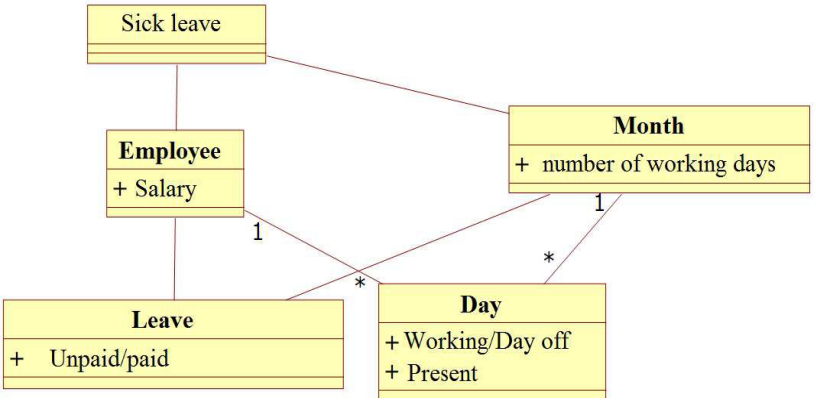

Fig. 3. Adding holiday and sick leave into model

After such integration, the model has become fairly complicated and complex. Besides, there is a question "How can we find out if the employee was at his work place or on his holiday in that day?" The business experts

reply "From a working hours register". We did not have this item, but its appearance changes the model:

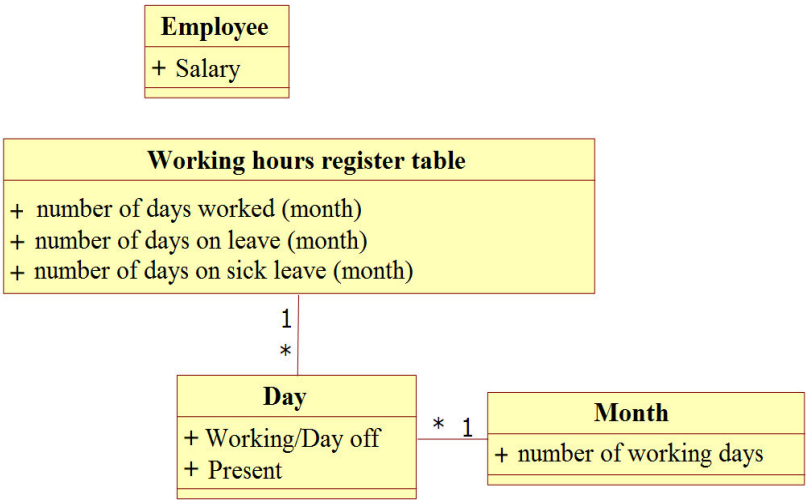

Fig. 4. Model after addition of working hours register

As we see, in the process of knowledge processing the initial model has changed essentially. Moreover, at the end we obtained reasonably flexible and understandable scheme similar to the scheme of behavior in the real world.

The domain model may and shall serve as a foundation for the common language (**Universal language**) for communication within the scope of the project on software development.

The model is a number of concepts which exist in the heads of project developers along with the names (terms), interrelations and interconnections which reflect their understanding of the subject. The terms and interconnections form semantics of the language, which is adapted for a domain area, but is at the same time fairly accurate for development technical needs.

The universal language shall become a main communicative tool within the project. Knowledge of the universal language is compulsory for IT-specialists, as well as for business experts. Owing to the use of the universal language and universal model we achieve:

- Verification of statements by business experts.
- General understanding of business requirements.
- Discussion of the model by the business and IT, search of balance in complex solutions.

- Business represents potential capabilities of the system and complexity of various updates.
- At operational stage, effective communication of business-users and IT without qualified translators-analysts.

Perhaps, the most difficult point of DDD is Model Driven Development (MDD). MDD is an architectural design with maximum exact correspondence between certain subset of a program and model elements. As it was described above, the model is a consequence of complex knowledge processing that leads the model to be an "ideal" of the domain understanding. But, the model becomes completely useless if it is not used to realize the program, it becomes abstract, bureaucratic, and unnecessary step which requires time, and as a consequence, loses its topicality. For this reason, DDD requires strict model following at software code realization. It is not always easy since software code development imposes many technical constrains. Thus, technical solutions that will be applied shall also be taken into consideration, and they shall be compulsorily entered into the universal language. Owing to this, the model and the universal language become more useful, since business experts start to operate with developers conceptions and understand technical part of the project deeper, that makes it easier to understand the project for all developing team [3].

For example, there is no concept of a "network link". But in the process of development of accounting software, this concept becomes very important, and experts in this field shall know about the existence of this concept, and it shall be taken into account while developing the model.

When designing by the model, it is important to use exactly the model terms in code realization. It is important for classes and models to be similar to those described in the model, and all technical details will be encapsulated.

In conclusion, we would like to note the following. DDD is not a linear process. In the process of knowledge processing, model driven design, and development of software and architectural solutions, new facts and knowledge may arise, the object domain may widen and contract, etc. This all leads to modification of the model and further to code refactoring. That, as a result, allows us to maintain topicality and usefulness of the project. Hence, the process of Knowledge processing – Model design – Advanced refactoring of the code is cyclic and constant. New knowledge shall lead to compulsory modifications in the code every time, and these modifications shall completely conform to the model [2, p. 23].

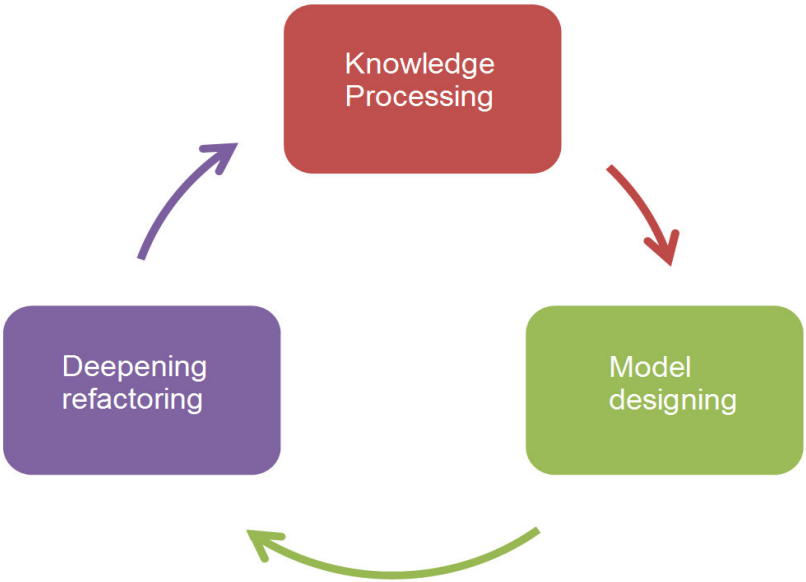

Fig. 5. Process interation

Except for all advantages and nice things, DDD presents certain difficulties. First of all, this understands of methodology itself and experience of work with it. Moreover, there are not many specialists, who have experience in development with DDD use on the market. Besides that, the methodology makes both developers and project member change the way of thinking. Additional specialists who will design the model are required. Furthermore, time for modeling is added. But all these shortcomings are leveled by the advantages of understanding of the project and its flexibility.

Conclusion

DDD is one of the most progressive methodologies of big project design. It allows us to achieve understanding between the system development members that is the basis for flexible, viable project, and, as a result, for achievement of the business aims.

However, application of this methodology is impossible without observance of the following terms:

1. Implicit acceptation of DDD by all project members, and application of its principles at all stages of the project life cycle.
2. Availability of experienced specialist in DDD.
3. Availability of business experts.

4. Compulsory strict development culture on the basis of the latest applicable methodologies (such as TDD (Test-driven development)).

1. Evans, Eric. Domain Driven Design (DDD). Structuring of Complex Software Systems, Williams 2010, p. 67-82. **2.** Grebvev, Nikolai. Domain Driven Design – How, Why and What For?: International scientific&practical conference "Application Developer Days", 2011. **3.** Nilsson, Jimmy. Applying Domain Driven Design and Patterns. Domain **4.** Driven Design with examples for C# and .NET, Williams 2008, p. 71-86. **5.** Fowler, Martin. Domain-Specific Languages, Williams, 2011. **6.** Tsepkov, Maksim. DDD: implementing the Tower of Babel project: International scientific&practical conference "Software People 2012", 2012. **7.** Fowler, Martin. Patterns of Enterprise Application Architecture, Williams, 2007.

Рецензент: д.е.н., професор Сазонець І. Л. (НУВГП)

Бузика В. В., технічний директор (ТОВ «Сігейтор»,
м. Дніпропетровськ)

DOMAIN DRIVEN DESIGN ЯК ОСНОВА ДЛЯ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

В цій статті розглядається проблема проектування складних обчислювальних систем з використанням методології предметно-орієнтованого проектування (domain-driven design). Викладені основні труднощі, з якими зустрічаються розробники програмного забезпечення и способи їх вирішення в контексті даної методології. В ній також розглянуті основні проблеми використання domain-driven design, випадки, коли ця методологія незастосовна або недоцільна.
***Ключові слова*: domain-driven design, предметно-орієнтоване проектування, модель, універсальна мова, об'єктна-орієнтоване програмування.**

Бузыка В. В., технический директор (ООО «Сигейтер»,
г. Днепропетровск)

DOMAIN DRIVEN DESIGN КАК ОСНОВА ДЛЯ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В данной статье рассматривается проблема проектирования сложных вычислительных систем с использованием методологии предметно-ориентированного проектирования (domain-driven design). Изложены основные трудности, с которыми сталкиваются разработчики программного обеспечения и способы их решения в контексте предлагаемой методологии. В нем также рассматриваются основные проблемы использования domain-driven design случаи, когда эта методика не применима или неуместна.
***Ключевые слова*: domain-driven design, предметно-ориентированное проектирование, модель, универсальный язык, объектно-ориентированное программирование.**