Domain Driven Development and Feature Driven Development for Development of Decision Support Systems

Conference Paper *in* Communications in Computer and Information Science · September 2012

DOI: 10.1007/978-3-642-33308-8_16

CITATIONS
2

READS
1,474

2 authors:

Paulius Danėnas
Kaunas University of Technology
30 PUBLICATIONS **243** CITATIONS
SEE PROFILE

Gintautas Garšva
Vilnius University
35 PUBLICATIONS **307** CITATIONS
SEE PROFILE

Domain Driven Development and Feature Driven Development for development of credit risk evaluation DSS

Paulius Danenas[1] , Gintautas Garsva[1]

[1] Department of Informatics, Kaunas Faculty, Vilnius University, Muitines St. 8, LT- 44280 Kaunas, Lithuania
{paulius.danenas, gintautas.garsva}@khf.vu.lt

Abstract. This paper describes adoption of Domain Driven Design and Feature Driven Development paradigms for decision support systems, partially for credit risk evaluation DSS case. Possible development scenarios using both of these methodologies are discussed, using transformations from previously described development framework. It is concluded that these techniques might be adopted for development of complex DSS.

Keywords: Domain Driven Design, Domain Driven Development, Feature Driven Development, credit risk, decision support system.

1 Introduction

Decision support systems are a tool that enables automation of credit risk evaluation process by integrating models developed using statistical, mathematical and artificial intelligence based techniques into an environment that is acceptable and convenient to use for credit manager. Such systems benefit banks and financial institutions by helping to examine and evaluate financial situation of such companies, extracting and providing them structured, detailed and visualized information, and thus supporting credit risk management process. The development of such systems is a sophisticated task as it involves development of specific modules and components. This also requires a lot of expertise from various experts which means their direct involvement into software engineering process which highly increases the complexity of design and development of such software.

Different methodologies have been developed for complex systems development. Domain-driven design and development, an approach extending model driven development has been introduced by Evans [1] and has been widely applied in industrial software engineering. MDA has been applied in development of intelligent systems, for e.g., Megableh and Barrett used their own developed methodology based on MDA for Self-adaptive application for indoor way finding for individuals with cognitive impairments [2]. Mohagheghi and Dehlen [3] developed a survey on applications of model driven engineering in industry which include examples on business applications and financial domain, as well as applications on telecommunications and other domains. Domain driven development for complex systems has been researched less extensively yet there are papers describing its application for complex systems development, for e.g., Burgstaller et al. [4] used domain driven development for monitoring of distributed systems.

However, currently there is lack of research targeted at designing complex decision support systems, especially using object-oriented techniques. The structure of DSS and expert systems in financial and credit risk field has been discussed and proposed in various papers [5-9], as well as agent-based system expert system development[10], yet they only describe high level structure and main components. Zhang et al. [9] present their framework of DSS structured as multilayer system, consisting of information integrated platform layer, utilization layer and information representation layer. This is similar to the framework proposed by the authors in [11]. Thus this paper tries to adopt and explore modern software design and engineering methodologies such as Domain Driven Design and Feature Driven Design in the context of sophisticated DSS framework for credit risk evaluation showing that such approaches might be a proper choice for such systems.

2 Domain Driven Design and Feature Driven Development – main concepts

2.1 Domain Driven Design

Domain Driven Design (abbr. as DDD; Evans, 2003) is based on such core concepts as domain, model, ubiquitous language and context. Only basic definitions of artifacts', patterns and concepts are given in this section; for more information of refer to [1].

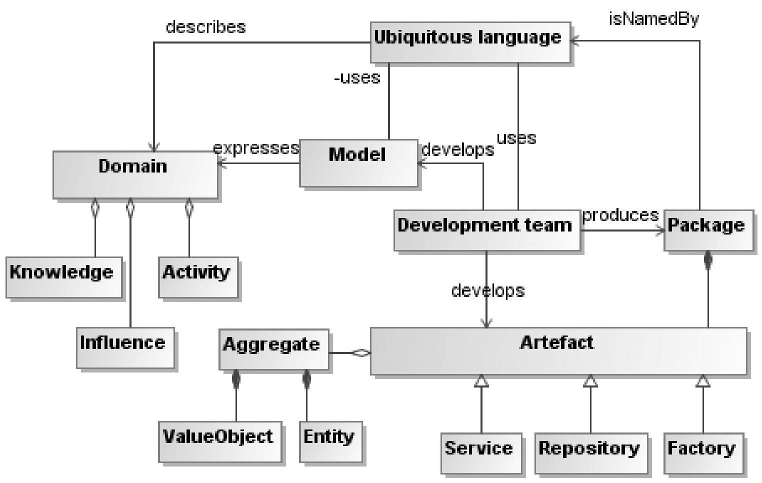

Fig.1. Concepts of Domain Driven Design

Domain can be described the subject area of application which can be expressed by ontology, as an influence or activity. The model represents a system of abstractions that describes selected aspects of a domain and can be used to solve problems related to that domain. Ubiquitous language is used by members of development team and helps to express domain model as well as connect all the activities of the team with the software. Context is described as the setting in which a word or statement appears that determines its meaning.

Evans also defines several artifacts to express, create, and retrieve domain models. Fig. 1 shows these artifacts in the context of overall DDD core model, as well as connections between main concepts and artifacts. Table 1 gives the basic definitions of these artifacts together with descriptions of core DDD artifacts.

Table 1. Core artefacts of DDD

DDD concept/model	*Purpose and description*
Entity	An abstraction which describes an object or a group of objects a thread of identity that runs through time and often across distinct representations. The objects are not defined primarily by their attributes; entity object might be matched although the attributes might differ.
Value object	Conversely from entity objects, this type of object has no conceptual identity but contains attributes
Aggregates	A set of value objects/entities connected by a root entity (aggregate root). Control all access to the objects inside the boundary is controlled through the root entity.
Service	Services are used when some concepts from the domain commonly are not modeled as objects, i.e., either distorts the definition of a model-based object or adds meaningless artificial objects (Evans, 2003).
Repository	An object that represents storage of domain objects, allowing to easily change storage implementations. According to Evans, they present clients with a simple model for obtaining persistent objects and managing their life cycle and separate application and domain design from data source or persistence technology.
Factory	DDD proposes extensive usage of OO Factory pattern for creating domain objects which allows to hide complex initialization logic and eliminates to requirement to reference the concrete classes of the objects thus allowing easy switching between several different implementations.
Packages (modules)	Represents separate aspects of implementation reflecting domain. This helps separate the domain layer from other parts of the implementation.

Although single model is an ideal solution, in development practice it is usually split into several models. Strategic Domain Driven Design provides a set of principles to ensure model integrity, refine domain model and work with several models. Evans describes such principles as bounded context, continuous integration, context map, shared kernel, conformity, anticorruption layer, open host service. These context driven patterns describe various aspects of both intersection and refinement for different models and contexts as well as uniformity and integration. For example, bounded context proposes that various contexts should be clearly defined and bounded whereas context map defines interactions between different models together with context boundaries and constraints. Anticorruption layer pattern offers an extensive usage of intermediate layers which act as mediators between different models or systems through interfaces. Thus no modifications are necessary for other systems which might represent different domains or subsets of these domains. Shared kernel facilitates usage of core subsets of each domain which might not be changed independently during the design by each counterpart (development team).

Among these principles, principles of distillation for separating the components and refactoring domain models were also proposed. This includes core domain, defining the most valuable and essential specialized concepts, generic subdomains which are essential for the full definition of the model and to the system, highlighted, segregated and abstract core, cohesive mechanisms as well as patterns for large scale structures, such as evolving order, responsibility layers, knowledge level and pluggable component framework. The core patterns propose different ways to extract domain core – highlighted core offers this by simplifying core domain description and highlighting its main aspects whereas segregated and abstract core models propose refactoring of core domain in order to obtain clearer models. Responsibility layers offer an interesting system engineering viewpoint as it proposes to refactor the model in a multilayered way such that the responsibilities of each domain object, aggregate and module fit neatly within the responsibility of one layer [1]. Knowledge level pattern extends this approach by implementing such layer system in hierarchical manner, where each level directly depends on lower level. This is an important aspect for design and development of complex decision support systems as they usually include several different domains and aspects which require different kinds of expertise (financial, statistical, machine learning, data source integration); therefore, different experts from various fields are required for development of such system. Pluggable component framework allows uncomplicated substitution of various implementations implementing corresponding interfaces; this can be seen as one of popular component-based development patterns. Such pattern allows separating and encapsulating several bounded contexts by exposing their functionality as components, with shared kernel as core.

Therefore, as Evans highlights, context, distillation, and large-scale structure design principles are complementary and interact in many ways, for e.g., a large-scale structure can exist within one bounded context, or it can cut across many of them and organize the context map. This will be shown later by a case study of credit risk evaluation DSS.

2.2 Feature Driven Development

Feature-Driven Development (abbr. as FDD) [13] is an iterative and incrementing model-driven software development process, based on short iterations and five activities: overall model development, feature list building, plan by feature, design by feature and build by feature. First two activities are general activities, which define general model and its structure. Other three activities are iterative, where special feature is developed. Palmer and Felsing [13] define feature as small function valued by client, expressed in such form: <action><result><object> (e.g., calculate rating of customer).

Each feature is designed and developed using model-driven methodology, specifically UML diagrams. Class, activity and collaboration diagrams (or sets of such diagrams) are usually produced for each feature, as well as for overall model; these models are refined during further development. An exceptional step is domain analysis for each feature done by domain expert.

FDD process usually uses modular structure which defines how various layers communicate with each other; in case of distributed system this architecture also describes inner communication of components. System separation into several layers is useful as it allows forming development teams for different components of such system by their skills, as well as defining interfaces for integration of these components. According to Palmer and Felsing [13], there are 4 layers in their proposed FDD architecture:

- User Interface (UI) layer, which interacts with Problem Domain layer, providing data views and enable users to easily invoke the problem domain features they need to get their desired results. Sub-layers, such as Navigation and Look and Feel sub-layer and Presentation sub-layer are also defined; the first defines how data for problem domain is presented and entered, invokes system functionality, related to problem domain, or navigates on the UI. Presentation sub-layer connects problem domain objects to corresponding objects in the user interface, the management of the user session, and the population of data for the various graphical elements in the user interface; Model-View-Controller pattern can be given as an example of such sub-layer implementation [13].
- Problem Domain (PD) layer – the most stable and independent layer which represents problem domain together with its inner logic and objects. Its development is the most important and therefore time consuming task in FDD architecture development as other layers directly depend on this layer.
- System Interaction (SI) layer – this layer allows to expose problem domain functionality for the external systems and components as well as use their functionality as components or Web Services. Main features which are implemented for this layer are subset of feature list features related with integration or interfacing of such components.
- Data Management (DM) layer, which is responsible of management of business objects. Its implementation might change according to changes in data storage infrastructure, thus it is recommended to separate problem domain and storage logic which is specific for particular implementation. Object-oriented persistence layer and mechanisms such as inner mappings between business objects and physical storage between are usually used to implement this layer.

3 DSS structure expressed in terms of DDD

This section describes integration and application of DDD for development of sophisticated DSS for credit risk domain consisting of several layers. DSS framework described in earlier work of the authors [11] is used as an case study. Their DSS is also described as multilayered structure which makes it easier to integrate and apply DDD by describing mappings between layers of credit risk DSS and corresponding layers in responsibility layer structure of DDD.

The proposed system framework consists of several layers. Only main points such as layer structure is presented here; more information can be found on original paper. The system supports intelligent and statistical model development, testing and prediction operations, as well as data analysis using statistical, financial and visual analysis techniques. Credit risk analysis can be viewed as an aggregation of these three as all these activities are present in this process.

The system consists of three core layers which represent each of domains related to the DSS:

- SVM-ML (SVM based machine learning) layer representing machine learning techniques and algorithms for model development, information processing, feature and instance selection;
- Data layer (DL) layer which defines data needed for modeling storage facility such as financial, operational, management, market, statistical, historic and macroeconomic data as well as metadata, such as reference or multilanguage data. Intelligent and statistical model repository together with metadata and execution log is also defined in this layer. The structure of the data is conformant to Basel II requirements as data representing both credit risk, operational risk and market risk is included in this layer.
- Credit risk evaluation layer (CRE layer) represents analysis, modeling, forecasting, visualization and evaluation business logic. Financial Analysis, Modeling and Forecasting modules implement analytics, simulations and forecasting of particular domain.

Such structure separates logic that belongs to different domains and enables reuse of developed components in other intelligent systems where similar problems are solved. Functionality of various supporting services is defined in additional sublayers, which belong to one or more of main three layers:

- Data source interaction layer – it is defined in both SVM-ML and CRE layers. SVM-ML layer interaction sublayer includes database interaction layer with object persistence and database connection frameworks, as well as various data standards including interoperable Predictive Model Markup Language (PMML) standard. It also defines the interfaces for information retrieval using Web Services or intelligent agents. CRE layer extends this layer with financial standards and data sources specifically for finance or credit risk related tasks (e.g., rating information). It also has a mapping package that contains the mappings between XBRL (and other standards) and data stored in Data Layer; structure of such mappings has been discussed in [4].
- Information Processing layer – it is also defined in both SVM-ML and CRE layers. This layer defines support for data preprocessing tasks, such as data retrieval, extraction and cleansing, normalization/standardization, imputation, transformation using factor analysis/PCA and other algorithms. The same layer defined in CRE layer defines tasks specific for financial domain, e.g., specific transformations, data transformation to absolute changes or changes expressed in proportional manner between particular ratios during particular period and etc.

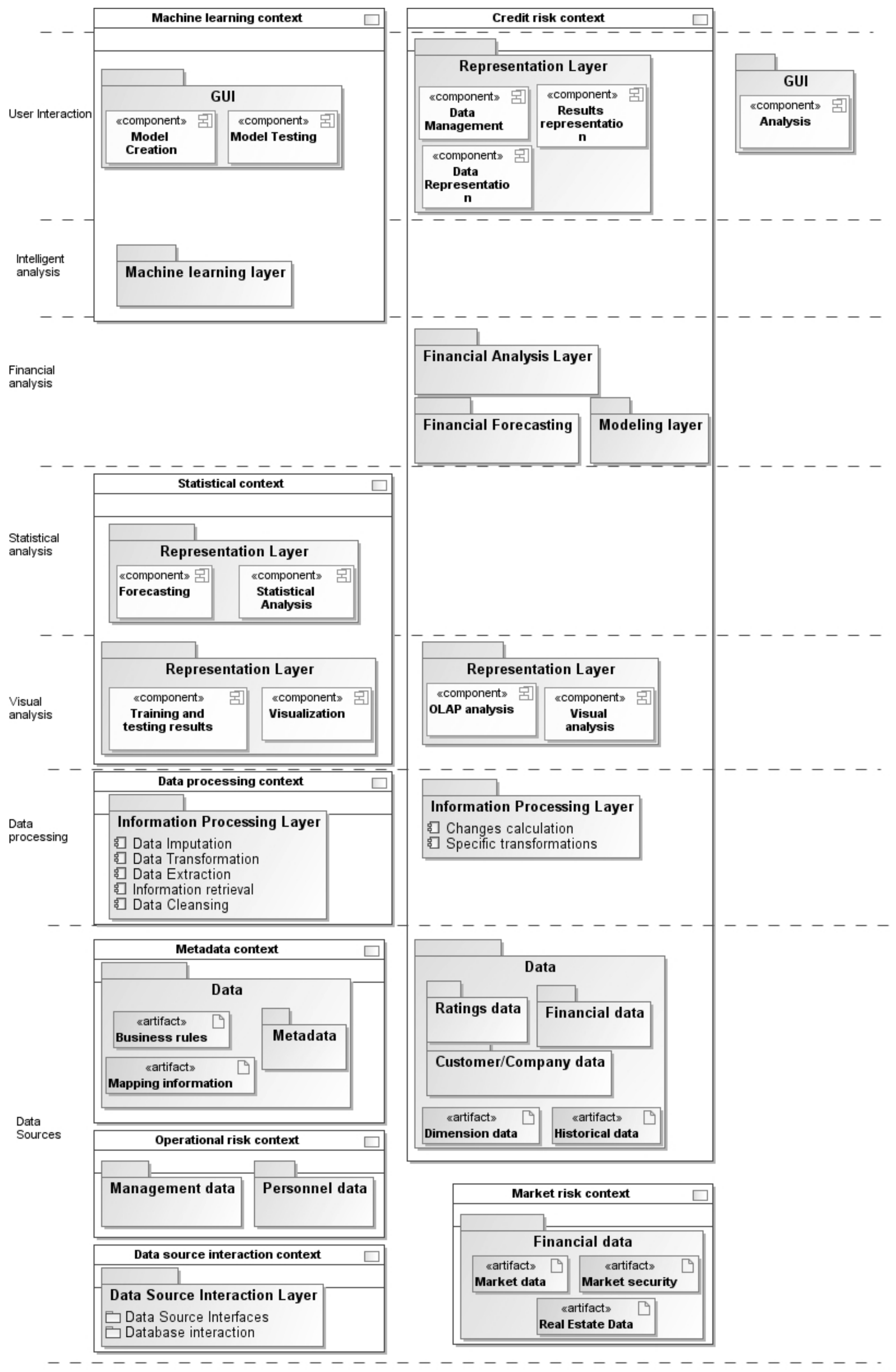

Fig. 2. Responsibility layers for credit risk DSS

- Representation layer – describes the logic used for representation and visualization of results. It is defined for both SVM-ML and CR layer. In case of SVM-ML this layer defines standard representations of training, testing and prediction results as well as their visualizations. CRE layer defines more sophisticated modules such as OLAP analysis, representation of financial analysis, simulation/modeling and forecasting as well as data management and representation functionality.

The use of such structure clearly enables application of multicontext approach therefore DDD can be defined as a good option for such system development. Several contexts can be distinguished:

- Statistical context;
- Machine learning context;
- Credit risk context;
- Operational risk context;
- Market risk context;
- Data source interaction context;
- Metadata context.

Each of these contexts represents different aspects of the system. This should not be confused with modules, as bounded contexts provide the logical frame inside of which the model evolves, and modules are more of a tool for organization of models' elements. To extend the framework by introducing additional layers with meaningful semantics, it was extended with 7 responsibility layers from DDD. These layers represent main software engineering and problem domains related to DSS development:

- User interaction – this layer represents relationships of components related to GUI development. This includes all functionality, related to data and modeling results presentation and visualization. Note that visual analysis also is also included in Visual analysis layer as various algorithmic techniques such as Self Organizing Maps or Principal Components might be used for visualization.
- Intelligent analysis – defines responsibilities for development of machine learning based techniques and methods with their integration into credit risk modeling process;
- Financial analysis – describes functionality necessary for financial analysis, such as financial modeling and forecasting;
- Statistical analysis – similarly to financial analysis layer, describes functionality necessary for statistical analysis;
- Visual analysis – here it is defined as representing functionality needed to visually represent modeling results of other analysis layers (intelligent, financial and statistical). Differently from User interaction layer, it is mapped to algorithmic functionality;
- Data processing layer describes implementations of methods for data preprocessing to perform model training and testing tasks;
- Data Sources layer represents data which is needed to perform modeling, analysis and decision support tasks, as well as their interfacing functionality. It includes data for credit, operational, and market risk analysis together with metadata and model repository. This layer describes the responsibilities related to development of data storage implementations.

Avram & Marinescu [12] state that common architectural solution for domain-driven designs contain only four conceptual layers. This is more practical for development, as the complexity of layers in Fig. 2 is simplified to four layers commonly used in almost every system. The simplified architecture (without bounded contexts described before) is shown in Fig. 3.

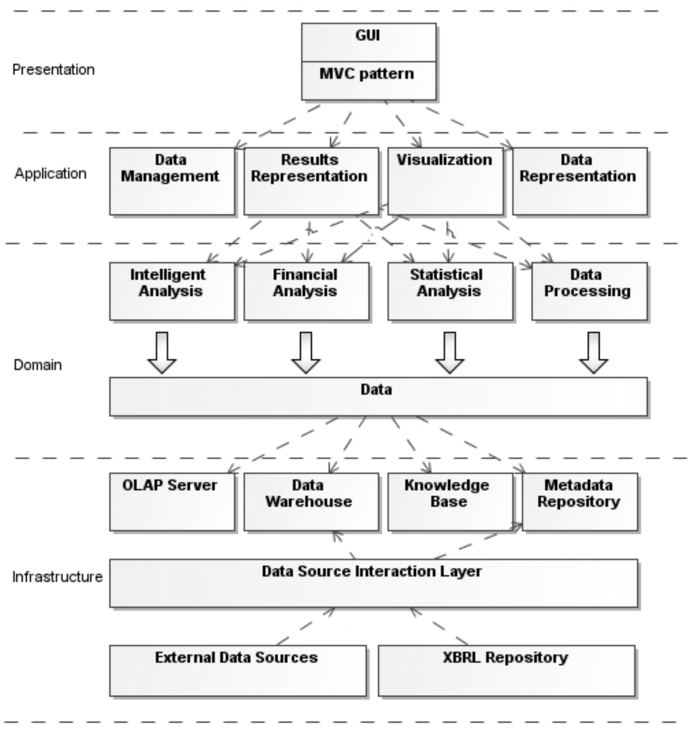

Fig 3. DSS framework consisting of 4 layers

The main difference is that analysis layers and data processing layer were into single joined domain layer; therefore, the data, used in risk analysis (i.e., defined in credit risk, market risk and operational risk contexts) is moved into Domain layer, while metadata and knowledge base (which also represents model repository) modules defined in Infrastructure layer, together with Data Source Interaction layer. Such distinction separates support, storage implementation and external data source connection functionality from problem domain which is directly dependent on data defined in Data Sources layer. These tasks are defined in Infrastructure layer; note that Infrastructure is defined in software level and is not dependable on the hardware. The presentation layer defines the same set of responsibilities and functionality as in extended model; however, Data Management and Data Representation activities were moved to Application layer, accordingly to MVC pattern. The domain layer is focused on core domain issues; therefore, it is not involved in infrastructure activities, such as management of metadata and external data sources or storage facilities.

.

4 Feature driven development adopted to proposed DSS

Feature driven development is also considered as a good choice for development of complex systems, as each feature is developed iteratively. This is especially useful if development and testing of some components is a sophisticated task requiring a lot of testing iterations; this is a good premise for development of intelligent systems which include machine learning and statistics based techniques. A feature can be represented as a class, component or a module. The technical architecture is described in Section 2.2; Fig. 4 presents its adaptation for credit risk DSS.

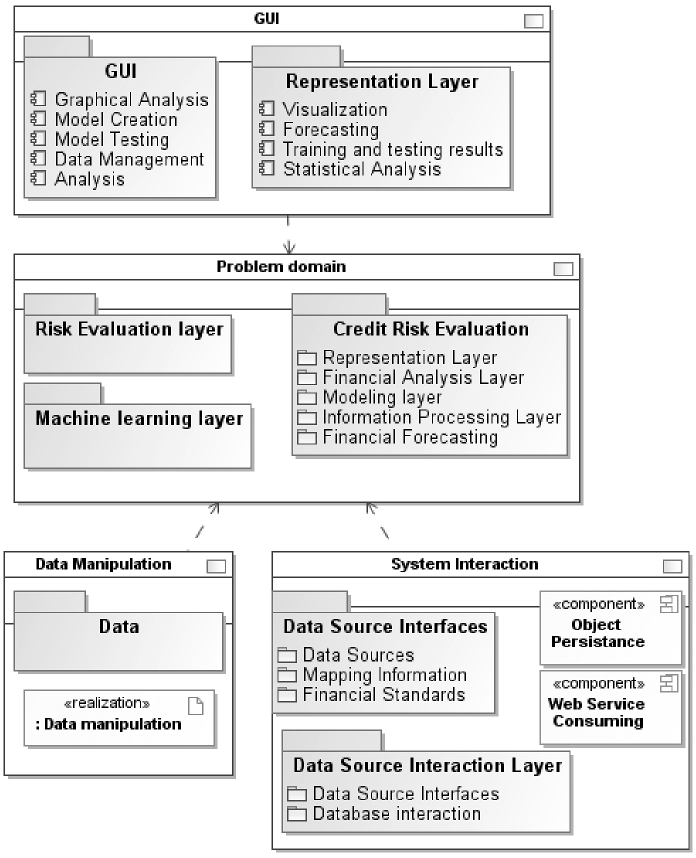

Fig. 4. DSS structure transformed into FDD technical structure

While comparing Fig. 3 and Fig. 4 it can be observed that the layer structure is very similar – both of these models have GUI and Problem domain layers, and System Interaction layer closely resembles Infrastructure layer in simplified DDD. The only difference is that, according to [12], Infrastructure layer contains all libraries and components for other layers (for e.g., persistence frameworks, web application frameworks etc.) as well as provides communication between layers, whereas System

Interaction layer is designed mainly for communication with external systems and Web Services. The integration of Web Services is an important criteria as authors propose implementation for credit risk DSS as distributed system which also increases its complexity. Data manipulation layer in FDD also closely corresponds to Application layer in simplified DDD model as they both represent Data manipulation functionality. Another difference is Data dependency in FDD Data Manipulation layer as, according to its definition, this layer is responsible for managing objects for Data as business objects whereas Data is mapped to Problem Domain layer in simplified DDD. However, this shows that FDD, similarly as DDD, can be applied for development of such systems. Note, that although separate features would be developed by separate development teams, integration of such features requires higher levels experts.

5 Conclusions

Since its development Domain-Driven Design and development was widely applied for software engineering, especially for complex systems. DDD focuses on understanding the customers' needs and the environment in which the customer works of techniques and aims to exploit the creativity factor in system development process. However, the analysis of its previous applications and adoptions for design and development of decision support systems showed that it is not widely used in this field. One of the main problems which arise during its adoption is the specific purpose of such systems which requires knowledge from different fields and different experts. Thus different approach from software engineering is needed to be taken. Yet, analysis of DDD patterns and techniques showed that it can be adopted for development of such complex intelligent systems. Usage of multiple contexts and layers helps to ensure proper coordination between several development teams which might involve experts from different domains. This paper analyses a case study of such complex system which involves extensive application of intelligent and statistical techniques for decision support in financial domain. It shows a transformation from previously defined DSS framework to a multilayered and multicontext structure. Another software engineering framework for such systems development, Feature Driven Development, is also discussed for the presented case. Although it does not offer full solution for software development (i.e., it does not comprise early project stages), it is merely concentrated on iterative development using Agile development principles and extensive MDA/UML. The analysis of its technical architecture and development capabilities showed that, similarly to DDD, it can also be adopted for development of complex DSS.

References

1. Evans, E.: Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison Wesley, 2003.

2. Magableh, B.; Barrett, S.; Self-adaptive application for indoor wayfinding for individuals with cognitive impairments. In: 24th International Symposium on Computer-Based Medical Systems (CBMS), pp. 1-6, Bristol (2011).
3. Mohagheghi, P. Dehlen, V.: Where Is the Proof? - A Review of Experiences from Applying MDE in Industry. In: Schieferdecker, I., Hartman, A. (eds.): Proceedings of the 4th European conference on Model Driven Architecture: Foundations and Applications (ECMDA-FA '08), pp. 432-443, Springer-Verlag, Berlin, Heidelberg (2008).
4. Burgstaller, R., Wuchner, E., Fiege, L., Becker, M., Fritz, Th.: Using Domain Driven Development for Monitoring Distributed Systems. In: ECMDA-FA 2005, LNCS, vol. 3748, pp. 19-24, Nuremberg, Germany (2005)
5. Zopounidis, C., Doumpos, M.: A preference disaggregation decision support system for financial classification problems. In: European Journal of Operational Research, vol. 130, pp. 402-413 (2001).
6. Cheng, H., Lu, Y.-C., Sheu, C.: An ontology-based business intelligence application in a financial knowledge management system. In: Expert Systems with Applications, vol. 36, no. 2, pp. 3614-3622, (2009).
7. Tsaih R., Liu Y-J., Liu W., Lien, Y.-L.: Credit scoring system for small business loans. Decision Support Systems, vol. 38(1), pp. 91-99 (2004).
8. Huai, W.: The Framework Design and Research On Enterprises Group Financial Decision Support System. In: Proceedings of Management and Service Science (MASS), pp. 1-4, Wuhan (2010).
9. Zhang, M., Gu, Y., Zhu, J.: Analysis of the Framework for Financial Decision Support System. In: Proceedings of 2009 International Conference on Wireless Networks and Information Systems, pp. 241-244, Shanghai (2009).
10. Guo-an, Y., Hong-bing, X., Chao, W.; Design and implementation of an agent-oriented expert system of loan risk evaluation. In: Proc. Of International Conference on Integration of Knowledge Intensive Multi-Agent Systems, pp. 41-45 (2003).
11. Danenas, P., Garsva, G.: SVM and XBRL based decision support system for credit risk evaluation. Proc. of the 17th International Conference on Information and Software Technologies (IT 2011), pp. 190-198, Technologija, Kaunas, Lithuania (2011).
12. Avram, A., Marinescu, F.: Domain-Driven Design Quickly. InfoQ, 2006, http://www.infoq.com/resource/minibooks/domain-driven-design-quickly/en/pdf/DomainDrivenDesignQuicklyOnline.pdf
13. Palmer S. R., Felsing, J. M.: A Practical Guide to Feature-Driven Development. Prentice Hall, 2002.