Using Domain-Driven Design to Evaluate Commercial Off-The-Shelf Software

Harald Wesenberg
Statoil ASA
Business Application Services – Energy Trading Solutions
Rotvoll
N-7005 Trondheim
+47 995 79 083
hwes@statoil.com

Einar Landre
Statoil ASA
Business Application Services – Application Development Center
Forusbeen 50
N-4035 Stavanger
+47 414 70 537
einla@statoil.com

Harald Rønneberg
Statoil ASA
Corporate Services – Information Technology
Forusbeen 50
N-4035 Stavanger
+47 915 76 165
haro@statoil.com

Abstract

Purchasing a Commercial-Off-The-Shelf (COTS) package solution can be a complex and daunting task. Selecting and evaluating the right candidate is difficult, especially when the solution aims at the heart of company business. The company's competitive edge must be maintained, while at the same time ensuring the intended goals such as reduced costs and better functional coverage. A good Enterprise Architecture should be a prime tool when evaluating several solutions against the company's needs.

In this paper we will recount the experience and lessons learned when we evaluated three COTS systems to replace a set of legacy oil trading and operations systems. Based on weaknesses in our Enterprise Architecture, we applied strategic domain-driven design principles to extend our Enterprise Architecture during the evaluation. We found that these techniques enabled us to thoroughly analyse our domain with the domain experts and provide answers based on tacit domain knowledge, without going through the cost and effort of performing a full-scale architectural analysis. At the same time, the tacit domain knowledge became explicit and shared, easing the communication with various stakeholders.

Categories and Subject Descriptors D.2.11 [**Software Engineering**]: Software Architectures

General Terms Management, Theory, Experimentation.

Keywords Domain-Driven design, Enterprise Architecture, context map, responsibility layer, information architecture.

1. Introduction

Statoil is in the process of replacing a set of legacy software systems with new systems supporting our Wet Supply Chain (WSC) [5]. One of the options we were considering was to buy a commercial off-the-shelf system to cover part of our needs.

After a Request For Information (RFI) had been sent out, responses were received from several vendors, and some of these were short-listed for further evaluation and a possible Request For Proposal (RFP). We planned to use the Enterprise Architecture for the WSC as one of the tools to assess how well each COTS candidate fitted into our overall architecture. We were especially interested in the following aspects of the candidates:

- **Functional Coverage:** How well did the candidates cover our functional needs
- **Information model:** Did the candidates have the necessary information properly structured to cover our information needs

Based on the architectural fit of the different candidates, we expected to select one or more of them for a RFP.

This evaluation effort coincided with our adoption of Domain-Driven design [3] and its use to expand the Enterprise Architecture for our existing system portfolio [5]. When we experienced problems with the use of our Enterprise Architecture, we decided to try using Domain-Driven design techniques to see if it could bring the evaluation further.

Before we continue the report, a short introduction to Enterprise Architecture and Domain-Driven design is in order.

1.1 Enterprise Architecture (EA)

According to [1] Enterprise Architecture (EA) identifies the main components of the organization, its information systems, the ways in which these components work together in order to achieve defined business objectives, and the way in which the information systems support the business processes of the organization. The components include staff, business processes, technology, information, financial and other resources.

Enterprise architecture is based on a holistic view rather than an application-by-application view. Most enterprises choose to do their Enterprise Architecture work according to the practices defined by available frameworks such as TAFIM [6], TOGAF [7] and Zachman [8] and tailored to reflect the architectural principles, standards and reference models defined by the individual enterprise. The frameworks provide a set of views supporting the different stakeholder interests, e.g. business process, information, functions and technical infrastructure.

OOPSLA'06 October 22–26, 2006, Portland, Oregon, USA.
ACM 1-59593-491-X/06/0010.

We have chosen to use the Enterprise Architecture as a foundation for describing the need for new IT systems and strategies for modernizing existing ones. It should provide a clear path for development or purchase of new systems and should be the natural start point when scoping and prioritizing new projects. For this to be possible it must be anchored in a joint business & IT vision identifying business requirements and IT objectives [2].

Although different frameworks have different perspectives, most of them have some common building blocks. These common building blocks of an Enterprise Architecture are found in Figure 1.

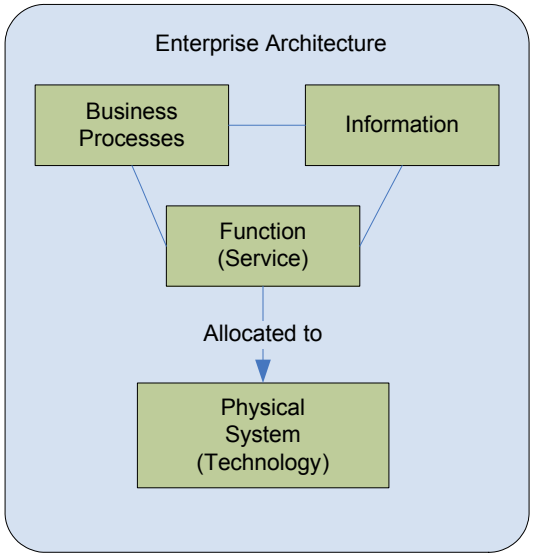

Figure 1: The classic building blocks of an Enterprise Architecture

1.2 Domain-Driven Design

Domain-Driven design is a philosophy whose focus is on the intricacies of the domain and where the objective is to make these intricacies explicit in the domain model and its implementation in code. According to [3] the premise of Domain-Driven design is two fold:

- For most software projects, the primary focus should be on the domain and domain logic and
- Complex domain designs should be based on a model.

Domain-driven design is not a technology or a methodology. It is a way of thinking and a set of priorities, aimed at accelerating software projects that have to deal with complicated domains. The primary source for these principles is Eric Evans' book on Domain-Driven Design [4].

Although Domain-Driven Design primarily is aimed at systems development and not COTS candidate evaluation, *context maps* and *responsibility layers* from the strategic level of domain-driven design proved to be very useful during our COTS evaluation.

1.2.1 Context Maps

Context maps are maps of the information system landscape partitioned into suitable groups of common functionality. Each context then contains information and functionality related to that context. Examples of a context are Physical Crude Oil Trading (information and functionality related to the process of trading physical crude oil) and Crude Supply Operations (information and functionality related to the operation of moving crude oil from location to another). Another usage of context maps is mapping out each information system in relation to other information systems. Examples of such contexts are the Trading system and the Supply Operation system from [5].

When the contexts have been identified, it is then possible to investigate the relationships between each context. Based on the characteristics of the relationship, potential challenges and interaction problems between the different contexts may be identified and described. The most common relationships are

- **Customer/Supplier:** One context is a supplier to the other, supplying information and functionality as required by the customer context. Agreement between the contexts are reached by negotiations.
- **Conformist:** The information model, functionality and architecture of conformist context must conform to the source context, without influence on how the context develops over time.
- **Shared Kernel:** Two contexts share a set of common information and functionality, and all updates to one context are also updates to the other.

Earlier we had performed various context map analysis on our existing portfolio of systems, and had found this to be a valuable tool in the communication of the inherent properties of our domain [5].

1.2.2 Responsibility Layers

During domain exploration, the resulting domain model often settles in *strata* (layers) where objects of similar use are grouped together. Examples of such strata we have found are

- **Capabilities:** Information and functionality related to determining and keeping track of what is possible within the given context. In our world, this could be stock levels, delivery obligations, pipeline infrastructure, vessel classifications etc.
- **Operational:** Information and functionality related to the performing of operational tasks. In our world, this is tasks such as trading activities, supply operations etc. Often, the operational tasks depend on information from the Capabilities layer.
- **Decision Support:** Information and functionality related to support the user in making decisions. In our world this is such activities as market exposure monitoring, risk control, supply planning. The Decision Support layer often depends on information from the Operational layer, and the results from this layer is often fed back to the Operational Layer as instructions for tasks to be performed.

Although these are three examples of responsibility layers in a domain, other layering will occur in other domains. The

importance lies in the recognition of this layering, and the resulting domain knowledge that was tacit, but now is explicit.

2. Use of the Enterprise Architecture

Using methods from a Scandinavian consultancy, we had previously developed an Enterprise Architecture for the WSC. Initially the architecture consisted of business, information and functional architectures, which had been extended with matrixes mapping elements of the different architectures together. The information architecture defines approximately 150 core information concepts grouped into 20 information groups, with the dominant information concept in each group used as group name. The functional architecture defines approximately 120 business functions grouped into 28 functional areas. Based on this, we then had a 20x28 matrix describing the high level information usage within the different functional areas. Similarly we had mapped information groups vs. business processes and function areas vs. business processes. An excerpt from such a matrix is shown in

Table 1 below.

Table 1 Excerpt from the Enterprise Architecture. A matrix is used to map information groups to function areas and show which function areas create/update (●) or read (○) which information groups.

Information Groups vs. Function Areas		Information Groups			
		Deal	Cargo	Position	Lifting Plan
Function Areas	Physical Trading	●	○		
	Supply Operations	○	●		○
	Supply Planning				●
	Derivatives Trading	●			
	Risk Control			●	
Legacy: Create/Update: ●, Read ○					

2.1 Granularity Problems

During our evaluation of the COTS candidates, we attempted to use our Enterprise Architecture as a tool to assess their ability to support the WSC, but soon found that the initial granularity of the information and solution architectures was too coarse.

One such example is the initial information concept Deal from the information architecture. As we worked with the different candidates, there were large differences in how the Deal concept was handled in each candidate. However, since we had only one Deal concept, the differences were not very visible in the Enterprise Architecture. We therefore split the initial Deal concept into several different information concepts: Physical Deal, and Derivative Deal. With these two new concepts, the differences between each candidate system became more visible. Similarly we had to split other concepts (e.g. Cargo, Voyage, Delivery) to illustrate the differences between the various candidates. As the granularity became finer, the matrixes and evaluations became unmanageable and incommunicable, and only the project core team was able to navigate and understand the vital information contained within the evaluation.

Another such example was the business function architecture. Similarly to the information architecture, the initial architecture had one functional area for physical trading. As we worked with the architecture and the COTS candidates we found that they differed significantly in their coverage of the functional area. However, we were not able to illustrate this using the functional architecture or the mappings to the business and information architectures.

2.2 Lack of Firm System Boundaries

Another problem facing us during the evaluation was the lack of firm system boundaries. Since the entire system portfolio was being considered, we could freely move functionality between proposed applications, so for each COTS candidate we could at will move function areas between applications and split function areas into new grouping of the functions according to the specific COTS candidate coverage. The Enterprise Architecture gave us no support in assessing the consequences of our work or deciding if one grouping was better than any other. This was due to the footprint resulting from each grouping across the different architectures and mapping matrixes.

2.3 Tacit vs. Explicit Knowledge

We felt that our Enterprise Architecture contained much implicit information based on the knowledge of the WSC architecture team. This tacit knowledge enabled us to make decisions on the allocation of functionality to the COTS candidate, but we were unable to share this tacit knowledge and thereby justifying the decisions we made. We had some experience with the use of strategic domain-driven design techniques, and wanted to see if this could make the information and knowledge more explicit.

3. Using Strategic Domain-Driven Design

Based on the identified weaknesses of our Enterprise Architecture, we felt that we had to expand it to be able to use it in the COTS evaluation. This would however require considerable time and effort, something we did not have at this stage of the project. Earlier we had used strategic Domain-Driven Design techniques to analyze our existing system portfolio [5] and based on that experience we wanted to try the same approach for this evaluation.

3.1 Exploring the Domain

We started by drawing a simple context map, shown below in Figure 2. This map grouped the different functional areas of the existing applications into contexts of related functionality. Since SAP-based functionality was not evaluated in this project, all application areas covered by our portfolio of SAP systems were contained in the SAP context. This was done to reduce the complexity of the diagram. These five contexts then formed the basis for all our analysis, and as the domain exploration and COTS candidate evaluation progressed, we regularly came back to these five contexts as a starting point for new forays into the domain.

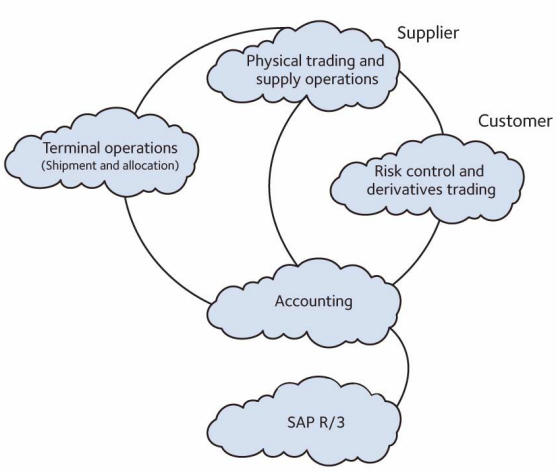

Figure 2: Simplified context map of application areas based on existing solution architecture. All application areas covered by SAP are contained in the SAP R/3 Finance and Accounting context.

Our first task was to get more detailed knowledge of the whole domain. We had several domain experts on different contexts within the domain, but few could claim extensive knowledge over several contexts. Thus, we ran a few workshops with the various experts, exploring the domain together over a couple of hours each time.

3.2 Using Responsibility Layers

Based on these workshops, we soon saw that the various functional areas from the solution architecture shared certain characteristics, and we started organizing the areas into responsibility layers based on these characteristics. Responsibility layers are a concept from the strategic Domain-Driven Design, and they are very useful as a tool for managing large portions of the domain. We found that organizing the application areas into such layers provided us with much knowledge about the build-up of the domain. In one such exercise shown in Figure 3 below we rearranged some functional areas from the solution architecture into three responsibility layers.

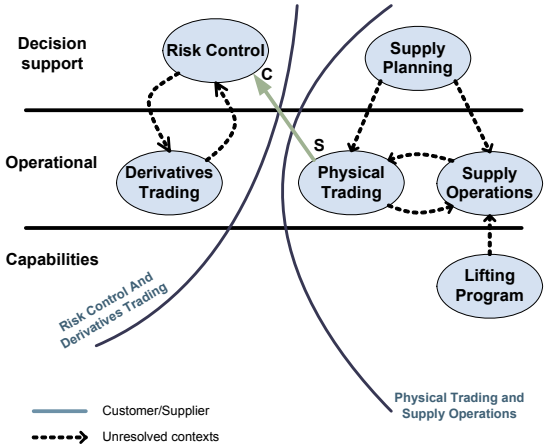

Figure 3: Responsibility layers for selected functional areas from the Enterprise Architecture. Used to explore the boundary between two contexts from Figure 2 (Risk Control and Derivatives Trading context vs. Physical Trading and Supply Operations context).

The responsibility layers gave us a clear picture of how some of our functional areas were dependent on each other and which areas should be kept together when deciding system boundaries. We identified three layers, Capabilities, Operational and Decision Support (the same three layers as in the introduction):

- **Capabilities** is the layer where our basic capabilities are laid out. In Figure 3 the Lifting Program is the functional area where all the available oil cargoes are received from the field operators. The lists of available cargoes are then used as basis for trading and supply operations. There is a context boundary towards the Terminal Operation context here, but this is not shown in this diagram.

- **Operational** is the layer where the day-to-day operational work is laid out. Trading, Supply Operations and Supply Planning are good examples of such functional areas.

- **Decision support** is where the system(s) aid the user(s) in their decision making process. An example is Risk Control, where various market risks are assessed, and decisions made on the basis of this assessment. The decisions are fed back to the operational layer as Derivatives Trading instructions.

When we had drawn our responsibility layer, we added the context boundaries from Figure 2. We saw that the interface between these two contexts was a simple one-way relationship, resulting in a non-complex interface with one-directional information flow and no shared functionality. Hence we found this to be a Customer/Supplier relationship (identified with C and S above), where Physical Trading supplies Risk Control with information necessary for Risk Control to fulfill its purpose.

We repeated this exercise several times until the application areas were mapped out and the context relationships were defined. In this example the application areas were in different responsibility layers, but this was not always the case.

4. Evaluating COTS candidates

After a period of domain exploration, we had a good overview of the various tricky aspects of our domain and felt ready to start analyzing our COTS candidates. For each candidate, we started out by sketching out a crude assessment of the context coverage, the result for candidate 1 shown in **Figure 4** below. Here, we saw that this COTS candidate had a fair amount of context coverage out-of-the-box, some possible extensions and some areas which could not be covered by the COTS system without major modifications.

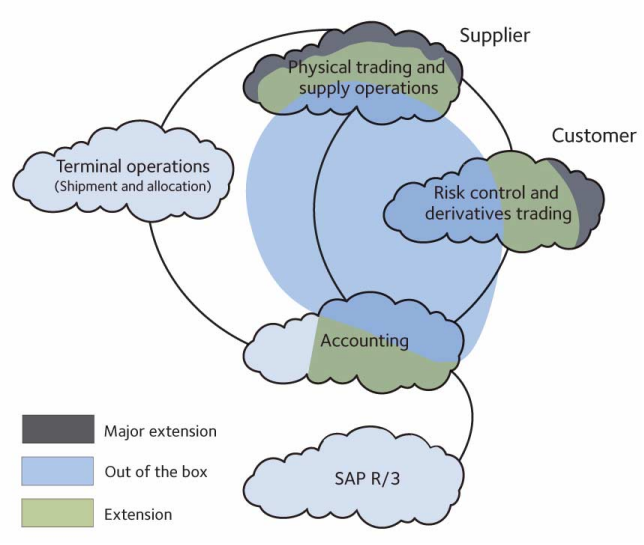

Figure 4: Analysis context coverage assessment for COTS candidate 1 showing how much is covered out of the box, what extensions are possible, and what cannot be covered without major modifications/extensions.

Based on earlier discussions with domain experts, elements of the Physical Trading and Supply Operations context (related to management of crude cargoes from the Norwegian Continental Shelf) had been identified as a core domain. A core domain is a context that supports the organizations core business, and it is vital to the competitive edge that this is efficiently supported by the processes and applications. We were particularly anxious to see the impact of the COTS candidates in this context.

During our analysis, we found that the analysis contexts we had previously identified were no longer correct based on the functionality covered by each COTS candidate and other systems. Hence, we abandoned our original analysis contexts and drew new, presumed contexts based on the particulars of each COTS candidate. For candidate 1, we identified several new contexts and divided the application areas between them. The new contexts are shown in **Figure 5** below. Based on our knowledge of the COTS candidate, we found that our needs for crude supply planning were poorly covered, and Crude Supply was introduced as a new context. Similarly, we found that some of our advanced risk management needs were not covered, and introduced Advanced Risk Management as a new context.

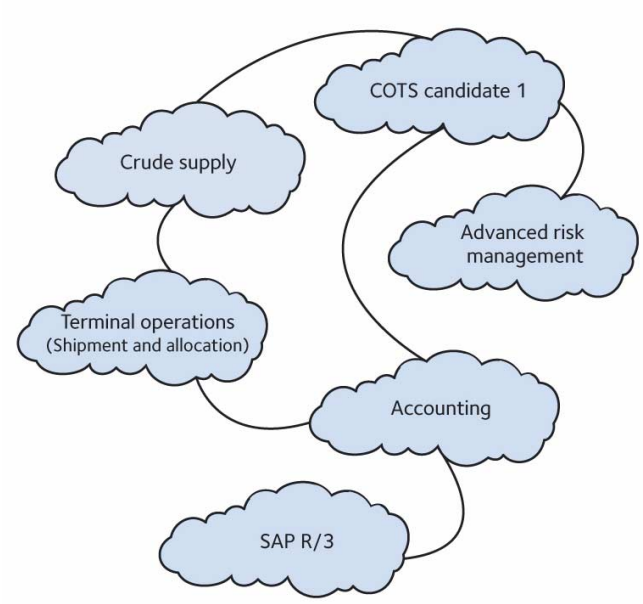

Figure 5: New, speculative contexts identified based on analysis of COTS candidate 1.

The context boundary between the COTS candidate and the Crude Supply context looked awkward to us, since we knew that a lot of the information and functionality needed for the COTS candidate would also be needed for the Crude Supply context. We decided to investigate this boundary further by using responsibility layers, and as we explored the domain we discovered that the two speculative contexts had a shared kernel as shown in **Figure 6** below.

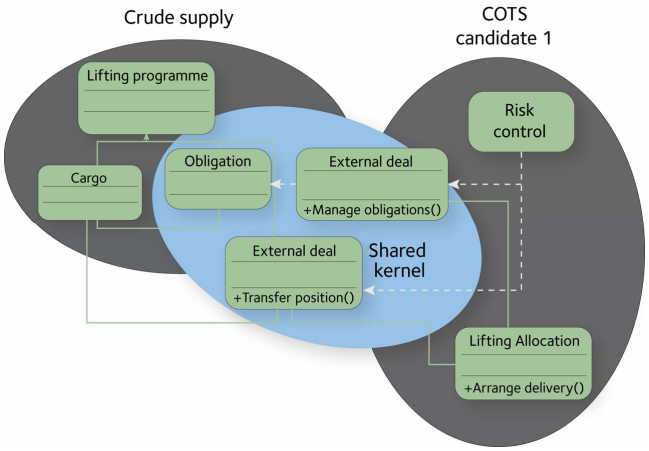

Figure 6: Sketched UML diagram showing three domain objects in a shared kernel between the Crude Supply Plan context and COTS Candidate 1. The Risk Control function area is shown as a package to show usage without cluttering with details.

A shared kernel is a set of information and functionality that is shared between two contexts. In this case, the shared kernel consisted of several domain objects that were necessary in both contexts. In our current systems, these domain objects are in the same system, and do not constitute a context boundary. Having system boundaries across this shared kernel would mean:

- Business logic would have to be developed in the Crude Supply context to duplicate functionality of the COTS candidate.
- The Crude Supply domain model would be dictated by the model of the COTS candidate.
- The cost of deviating from the model would mean construction of anti-corruption layers [4], in itself a costly and complex task.
- Integrating the two contexts would be expensive and demanding.
- The shared kernel could not be a service provided in one of the contexts, as both functionality and information were needed outwards in the different contexts.

The identification of a shared kernel between the Crude Supply context and the COTS candidate enabled the project group to focus the analysis efforts, investigating functional coverage and information needs in more detail. Based on the results of this analysis, we could state that this candidate would be unsuitable for us.

For the other COTS candidates, we produced the same context coverage assessment diagrams as shown in **Figure 4**. Based on these diagrams, we explored the domain as best we could based on assumed implementation contexts. This enabled us to assess how well each candidate fit into our overall Enterprise Architecture.

After a period of evaluating each candidate against our context maps, the project group reached the conclusion that none of the proposed COTS candidates fitted into our Enterprise Architecture. Therefore the project group recommended for the steering group that the new solution for the WSC would be based on custom development. The steering group accepted the recommendation, and custom development will be started in the fall of 2006.

5. Conclusion

Before we started this project, the stakeholders had a clear expectation that the Enterprise Architecture would be of help in evaluating the various COTS candidates. When we tried to use our Enterprise Architecture, we found that the architecture was not detailed enough, and that extending the architecture with more details would make it unmanageable with the tools we had at our disposal. The cost associated with such an effort would also be formidable, whereas the approach outlined in this paper enabled us to achieve good and accurate results over the course of a few workshops of a couple of hours each.

Our experience in this project shows us that extending our Enterprise Architecture with techniques from strategic level Domain-Driven design are useful to:

- **Facilitate domain exploration by use of a common language.** Strategic domain driven design gave us a common language with which we could talk about the domain. The use of clearly defined terms and concepts enabled the different project participants to understand and contribute to the exploration of the domain.
- **Focus on the core aspects of the domain.** A traditional Enterprise Architecture puts equal emphasis on all aspects of the domain. Using Domain-Driven design techniques, we were able to focus on the aspects of the domain that was vital to the task at hand and put less emphasis on the aspects that were not relevant.
- **Make tacit knowledge explicit.** By using responsibility layers and context maps, we were able to label relationships and dependencies in our domain, thereby making tacit knowledge from each project member explicit and shared with the others.
- **Investigate system boundaries.** By using speculative context maps based on each COTS candidate we were better able to investigate new system boundaries and determine whether integrating the different COTS candidate would be expensive and challenging.

In a traditional Enterprise Architecture this would of course also have been possible. The result would have taken considerable more effort in time and money, and be more dependent on the individual skills of the enterprise architects.

Acknowledgements

Our thanks go to Eric Evans for his helpful discussions with respect to the content of this article and his mentoring skills. We also thank Olaf Zimmerman and the ACM rehearsal staff for their help during the preparation of this paper.

References

[1] Armour, Kaisler, Y. Liu, A big picture look at Enterprise Architecture, IEEE IT Pro January/February 1999.

[2] Armour, Kaisler, Valivullah, Enterprise Architecting: Critical Problems, IEEE Proceedings of the 38 Hawaii International Conference on Systems Sciences – 2005.

[3] Domain-Driven Design, http://domaindrivendesign.org/.

[4] Evans, E., Domain-Driven Design, Tackling Complexity in the Heart of Software, 2003, Addison-Wesley, ISBN 0-321-12521-5.

[5] Landre E., Wesenberg H., Rønneberg H., Architectural improvement through use of strategic level Domain-Driven Design, OOPSLA 2006 practitioner report.

[6] TAFIM, http://www.sei.cmu.edu/str/descriptions/tafim.html/.

[7] TOGAF, http://www.opengroup.org/architecture/togaf/.

[8] John Zachman http://www.zifa.com/.