Architectural Improvement by use of Strategic Level Domain-Driven Design

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

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

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

Abstract

In this paper we present the experience gained and lessons learned when the IT department at Statoil ASA, a large Oil and Gas company in Norway, extended their Enterprise Architecture with strategic level Domain-Driven design techniques and used the extended Enterprise Architecture to improve the software architecture of a large enterprise system.

Traditionally, Enterprise Architecture has been prescribed as the key tool to conquer complexity and align IT development with business priorities and strategies, but we found our Enterprise Architecture too coarse to be practical useful at the software level.

By extending our Enterprise Architecture with context maps and the process of context mapping valuable insight was gained, insight that enabled better scoping of new projects and architectural improvement of existing software in a controlled way.

In addition, use of responsibility layers combined with context maps reduces the perceived complexity of the architecture. Use of other techniques such as distillation and identification of the core domain looks promising at the tactical level of a single project, but its value is more uncertain at the strategic level.

The key issue is that large enterprise systems do not have a single core. On the other hand, at the project level, there should always be a core, and the project is best of by knowing its core domain and aim its best resources to work with the core.

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

General Terms Design, Theory, Management

Keywords Domain-Driven design, enterprise architecture, context map, responsibility layer, complexity, distillation

1. Introduction

Statoil ASA has embraced Enterprise Architecture as one of its means to better align development of corporate IT systems with business priorities and strategies.

One of our pioneering areas for enterprise architecture adoption was the Wet Supply Chain (WSC). The WSC is the set of business processes that supports Statoils sales and delivery of crude oil, refinery products and Liquid Natural Gas (LNG) to internal and external customers. The WSC supports a global business operation that depends heavily on efficient IT tools.

The main cause driving the Enterprise Architecture effort in the Wet Supply Chain is the need to replace a set of large legacy systems with a combination of commercial packages [11] and custom made solutions. The replacement is motivated by new business requirements that can not be met within functional and technical architectures of the existing systems. The endeavour is organized as a program and the plan is to have made a complete replacement within a timeframe of three-to-five years. The first new systems were deployed for production in 2005.

In our attempt to develop and use the Enterprise Architecture for the Wet Supply Chain, we found that our Enterprise Architecture did not provide the tools needed to address key concerns when designing and integrating large scale software intensive systems. While enterprise architects focus on business processes, functions and information concepts, software architects have to focus on boundaries and interfaces.

As the work with our Enterprise Architecture correlated in time with our adoption of Domain-Driven design [3], we discovered that the strategic part of Domain-Driven design provided the needed mechanisms.

Our experience report is founded on practical work done in context of a software development project. The project objective was to replace paper based cargo folders with a digitized archive and follow-up capabilities.

Before we continue our experience report, a short introduction to Enterprise Architecture and Domain-Driven design is required.

1.1 Enterprise Architecture (EA)

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

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 Zachman [6], TOGAF [10] and TAFIM [9], tailored to reflect the architectural principles, standards and reference models defined by the individual enterprise. The frameworks typically provide an architectural lifecycle process and a set of views supporting the

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

different stakeholder interests: business process, information, functions and technical infrastructure [2].

The purpose of the Enterprise Architecture is to provide the foundation to describe the need for new IT systems and strategies for modernizing existing ones. It should provide a clear path for acquisition 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 and IT vision identifying business requirements and IT objectives [2].

Figure 1 illustrates common Enterprise Architecture building blocks.

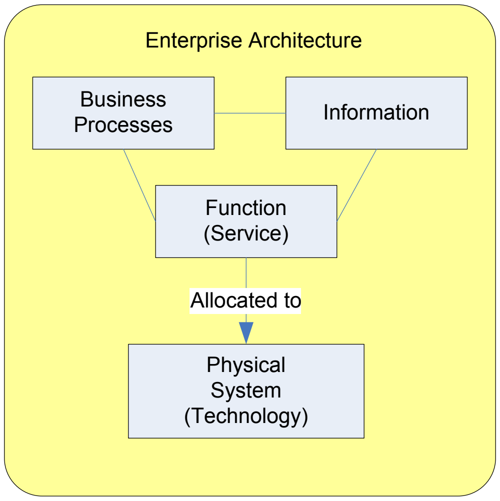

Figure 1. Enterprise architecture building blocks

1.2 Domain-Driven Design

Domain-Driven design is a philosophy whose focus is 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 [4] the premise of Domain-Driven design is two fold:

- For most software projects, the primary focus should be on the domain and domain logic.
- 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 [3]. Basically Domain-Driven design can be divided into three areas:

Basic building blocks – Addresses how the domain is separated from technology by use of a layered architecture, combined with practical object oriented design patterns.

Sophisticated models – Addresses how the software is aligned with domain expert thinking, domain concepts are made explicit in code and refactoring of the code is driven by domain insight.

Strategic design – Addresses model integrity and management of complexity in large systems. Strategic design provides three core building blocks:

- Context mapping
- Distillation
- Large scale structures

Of these building blocks context mapping is the important one that constitute the core of this report, while distillation and large scale structures provide useful architecting principles and guidelines that only will be briefly touched.

2. Context Mapping

A context map is a drawing that documents modelling contexts and their relationships. Large systems contains multiple modelling contexts, therefore we have depicted the modelling context of interests, not the applications or information systems that implement the different contexts.

The context map in figure 2 depicts the situation in the Digital Cargo File (DCF) project, with the project responsibilities on the right hand side of the figure.

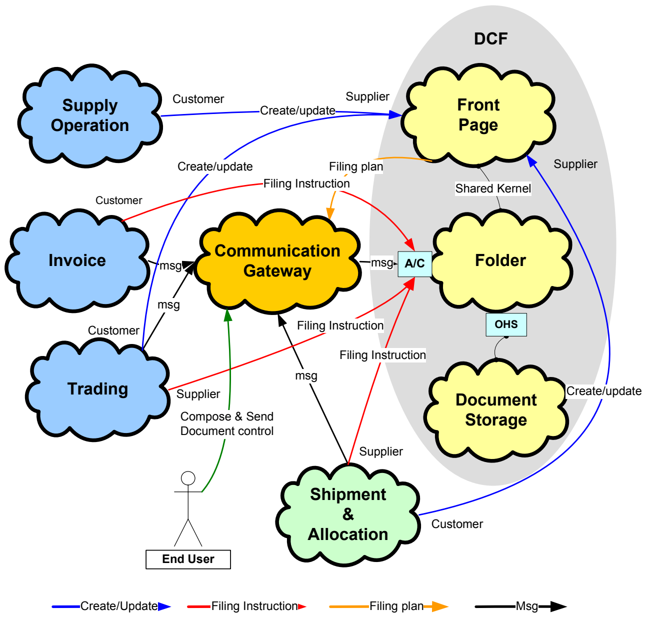

Figure 2. Context map – Digital Cargo File

On the left side of the context map we find three existing contexts: *Supply Operation* – supporting delivery of cargo, *Invoice* and *Trading* – supporting sales of cargos. These three contexts exist inside different legacy systems that must be integrated with DCF. These contexts were identified when we developed a context map for the whole WSC as part of our extended Enterprise Architecture effort [11].

In the middle we find the *Communication Gateway*, enabling the business to send and receive email, fax and telex. The end-user is represented to illustrate the fact that humans must operate the communication gateway directly for two purposes: 1) Manual sending and filing of business messages. 2) Manual filing of inbound messages from counterparties. This operation of filing messages is known as document control.

On the right side we find the DCF information system with its three distinct model contexts: *Front Page* providing the cover page of a physical folder – used for content follow-up and case management, *Folder* providing the logical storage model and *Document Storage* defining the physical document storage model. The purpose of the DCF information system is to digitize the process of handling unstructured information (email, fax, telex) associated with actual cargo and deals, and thereby improve the business operation. Before DCF unstructured information was printed and stored in physical folders.

At the bottom we find *Shipment & Allocation* supporting oil field operation and owner allocation of produced volumes.

2.1 Context relationships

In addition to the contexts the context map includes the actual relationship between contexts. Here is a short introduction to the relationships used in the map:

- **Customer / Supplier** means that the teams responsible for the two contexts have a customer / supplier relationship. Practically this implies that the supplier context must provide what ever is required to the customer context. This relationship is well regulated in terms of who is responsible for what [3].
- **Shared Kernel** means that the code base between the two context's are shared. A shard kernel means that changes made in one context most likely impact the other. Having a shared kernel without knowing it leads easily to undesired situations and defects. It is a take care type relationship [3].
- **Open Host Service (OHS)** means that a context provides access to its model using a defined service interface. The documentation of any published service should be based on the *published language* [3] pattern. The purpose of the published language is to define the translation of concepts between models. It could be claimed that the number of OHSs indicate how service oriented the architecture is.
- **Anticorruption Layer (A/C)** means that the context with the A/C layer attached protects itself from the context it is connected to. The purpose of the anticorruption layer is to translate between modelling contexts. Anti corruption layers are very similar to Open Host Service but it is used by the context who integrates with what to them is an alien model. Development and deployment of anticorruption layers involves use of application integration middleware, and are as such expensive modelling constructs [3].

For more relationship patterns and deeper descriptions consult the book [3]. One interesting aspect of context mapping is its alignment with well documented systems architecting heuristics [7] including: *Don't partition through regions where high rates of information exchange is required*, *Choosing the appropriate aggregation of functions is critical in systems design* and *The greatest leverage in systems architecting is at the interface*. These heuristics, among others, guided our analysis and the subsequent improvement process.

2.2 Context map analysis

With the context map in place it is time for analysis. The applied analysis technique is founded on Hitchins [5] complexity theory.

According to Hitchins is complexity subjective, where the perception of complexity is related to the combination of: variety, connectedness and disorder. Practically this mean that a system with low level of variety, high degree of order and low degree of connectedness is perceived less complex than a system with high degree of variety, lack of order and high degree of connectivity.

The project team faced a set of problems that was hard to understand, basically because the team was overwhelmed with details. They could not see the forest from the threes.

From analysis of the context map (Figure 2) it became clear that most of the encountered problems were caused by two factors:

- The role of the communication gateway
- The role of the front page

Both factors drive's the perceived complexity due to a high degree of connectivity and disorder. The actual problems will be discussed in the subsequent sections.

2.2.1 The role of the communication gateway

The communication gateway had been operational for years when the digitized archive was envisioned. Inbound and outbound messages were printed and filed into the correct cargo folder whose front page was updated to reflect the changed state of the folder. As a curiosity its worth mentioning that the folder filled the role as relay-stick used to pass the "case" from cargo operation to deal handling. Users found needed folders by inspecting each others desk, i.e. the folder fulfilled a role as a human workflow tool.

In addition to provide transmission of messages, the communication gateway was responsible for tracking received and sent messages. This was known as document control. Users interacted directly with the communication system. It was assumed that the introduction of a digitized archive should support the same work practice.

When the project started to integrate with the communication gateway it was overwhelmed by problems such as:

- The need to replicate filing information into the communication gateway for the purpose of filing. I.e. provide archive references.
- The need to extend the communication gateway with additional user interfaces for the purpose of filing manually sent and received messages into the archive.
- Complex interface and information flows between front-end systems (Trading, Operation), the communication gateway and the DCF.

In the context map most of these problems materialize as complex dependencies between contexts (Figure 2), i.e. the high level of connectivity.

2.2.2 The role of the front page

In the paper based cargo file system the front page represented the key tool for follow-up of the actual folder (case). The front page contained aggregated cargo and deal information originating from the Supply Operation and Trading contexts respectively, combined with relevant information about the communication with the actual counterparties. The actual information was written in hand on the physical front page of the folder.

Front page content combined with which desk the folder resided on provided the required workflow support in the paper based operation. When the paper based folder was removed the need for workflow support was still there, and the needed capabilities had to be provided by DCF.

The project decided to provide the required capabilities by implementing the front page concept as an integral part of the archive. The front page was updated automatically by Trading and

Supply Operation. Further the front page was extended with annotations to allow the users to use the front page as a scratchpad. The situation was further complicated as each business unit had their own variant of the front page but the project managed to convince the users to standardize on a limited number of variants. In the end there was one front page type for crude oil, one for refined products, one for liquid gas and one generic to support the remaining needs.

After deployment of the first version of DCF the business have requested more advanced case management and workflow capabilities. These requests indicate that the attempted implementation of the paper based front page concept did not provide the needed capabilities. Basically the users need a workflow system that monitors change and involves them when human attention is required.

2.2.3 Synthesis

With the context map in our hands, analysis of encountered problems turned out to be easier as the cause of problems became visible. They directly related to unhealthy structures in our architecture:

- Having the communication gateway as a spider in the web became counterproductive when moving from a paper based archive to a digital archive. The different links to the communication gateway represents only the top of the iceberg.
- Extending the archive with case management capabilities polluted the archive with functionality not related to its prime objective: filing and retrieval of documents.

Based on analysis of the current situation a new context map reflecting a to-be picture was established and later used to scope a new project. The new projects objective is improved software architecture, case management and document control capabilities. The improved context map was developed in workshops using a smart-board for efficient documentation of the process and its results.

2.3 Recommended changes

As stated in the previous section a new context map reflecting how we wanted our architecture to look like was developed. The key changes are:

- Front page concept is separated from the Folder context using Open Hosted Service and extended to provide more sophisticated workflow and case management capabilities.
- Document control moved from the Communication Gateway to the Folder context and the context is renamed to Folder and Document Control.
- Front end systems interact with the Folder context through the Filing Service.

The actual changes are illustrated in figure 3 and described in more detail in the subsequent sections. The effect of the suggested changes is reduced connectivity and thereby a less complex architecture compared with figure 2.

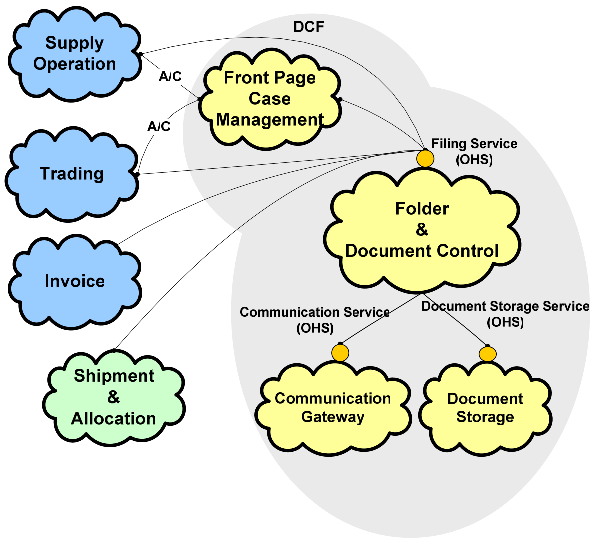

Figure 3. To-be context map

2.3.1 Front page and case management

Since the *Front Page* aggregates deal and cargo information that belongs to *Trading* and *Supply Operation* contexts for the purpose of case management, we decided to separate the *Front Page* context from the *Folder* context by introducing a service (Filing Service in figure 3).

With respect to *Supply Operation* and *Trading* the *Front Page* is protected from these two legacy contexts using anticorruption layers.

The *Front Page* interacts with *Folder & Document Control* through the Filing service (Figure. 3). This ensures loose coupling and facilitates a more service oriented architecture.

The main benefits from separating the *Front Page* from the *Folder & Document Control* are reduced coupling and more cohesive contexts with defined responsibility and interfaces.

2.3.2 Document control

Document control is the name of the capability enabling users to file inbound messages into the archive. Due to historical reasons document control was part of the responsibility of the *Communication Gateway*. By moving document control from *Communication Gateway* and merge it with the *Folde*r context two important improvements can be achieved:

- Users do not need to interact with the Communication Gateway improving their operational efficiency. Average time used to file an inbound message is estimated to be reduced by 30 seconds/message. With 1000 messages a day, this adds up to more than 8 working hours a day, time that can be spent on more productive activities than filing messages into the archive.
- Front end systems (Trading, Invoice, etc) do not need to interact with both the communication gateway and the folder contexts, reducing the number of couplings by 4, from 12 to 8.

The impacts of these changes are clearly visible in figure 3.

2.3.3 Filing and communications service

Domain-Driven design advocates a principle called *intention revealing interfaces*. In a context map such intention revealing service is expressed as an *Open Hosted Service* (OHS). The nature of an OHS is described in section 2.1 and more details can be found in [3].

The project decided to use an OHS as access point to the *Folder & Document Control* context. The benefit from an OHS is that the interface is documented and published as a first order design artefact that facilitates reuse and loose coupling.

The OHS pattern is in line with the principles of service oriented architecture (SOA) [8] and systems architecting heuristics with their focus on interfaces and boundaries [7].

2.4 Summary

We have now been through how context mapping and context relationships can be used to analyse and improve software architecture.

We have also seen that the actual contexts are derived from our extended Enterprise Architecture, and thereby turning the Enterprise Architecture into a useful tool for software architecture improvement.

Context mapping for model integrity represents the first leg in strategic level Domain-Driven design. In the next two sections *distillation* and *large scale structures* will be briefly touched.

3. Distillation

Distillation is about separating the important from the less important [3]. Ideally it should be possible to identify the problem area that motivates development of this actual software. That part of the domain is called the *core domain*. To be able to keep the core as small as possible, some domain related functionality should be moved out of the core, allowing us to let our most skilled people focus on the core [3].

This moved out part of the domain is called *supporting domain(s)*. That means it addresses domain specific concepts, but the required capabilities need only to be good enough. There is no need for a sophisticated model [3].

The last kind of software is the one that is required but does not address any domain specific knowledge at all. Such software is called *generic sub-domain(s)*. This software should, when practical, be based on commercially available packages [11] or open source offerings.

Applying distillation on the content of the context map in figure 3 the following story could be told:

- Front Page / Case Management enables the business to manage cargos and deals with respect to communication with shippers and counterparties, and supports the primary business processes related to cargo and deals within the Wet Supply Chain. In context of digital cargo files this makes it the *core domain*. It is these capabilities that justify the digital cargo file project.
- Folder & Document Control contain domain specific information as folders and sections reflect the way the business views documents in context of deals and cargos and tracks the state changes attached to documents sent and received from counterparties. This makes it a *supporting domain* in context of digital cargo files project.
- Communication Gateway and Document Storage does not contain any domain specific information, and the actual implementations are based on commercial available products. They are classical examples of *generic sub-domains*.

To understand the difference between the core, supporting and generic domains is critical as resources should be prioritized into development of the core, and supporting domains should be just good enough, potentially outsourced or procured. Improvements of supporting domains should be motivated from documented benefits in the core [3].

Experience indicate that distillation at the enterprise level is hard, why is trading more core than supply operation? On the other hand, there are concepts inside trading that are more crucial than others, and being able to find and prioritize these concepts sounds as a good idea. The effect of this discovery is that distillation seams to be more a tactical than strategic tool.

4. Large Scale Structures

Context maps are valuable tools to ensure model integrity at both system and project level. The challenge though is that for a large domain the context map itself become complex and unmanageable as the forest cannot be seen for the trees.

There are two elements from the large-scale structures that have proven valuable: the principle of evolving order and the use of responsibility layers.

Evolving order is a design philosophy founded on the fact that up-front designs, not based on experience, tend to fail. The message of evolving order is that conceptual large scale structures should evolve, and reflect our understanding of the domain at hand. The large scale structures should not constrain designs and model decisions that require detailed knowledge [3].

Responsibility layers address the need to handle large swaths of the domain in a coherent way. Its principles are derived from the architectural layering patterns, but applied on a more abstract level. The naming conventions applied for the different responsibility layers reflect the way we choose to think about the domain at hand [3].

Figure 4 illustrates how the context map from figure 3 can be reorganized into a layered representation. The placement of contexts in the different layers illustrates the responsibilities found in the WSC [11].

With the responsibility layers in place it might be easier to explain why the *Front Page* should be separated from the *Folder* context. The two contexts resides in different responsibility layers.

Another discussion that might originate from figure 4 is in which layer should hold the *Front Page* context. Is it part of Operation or Decision Support? We leave it for the principle of *evolving order* to sort out in the future.

The experience with use of responsibility layers from the COTS evaluation [11] indicate that use of responsibility layers simplifies communication with stake holders, because they reduce perceived complexity by introduction of more order [5].

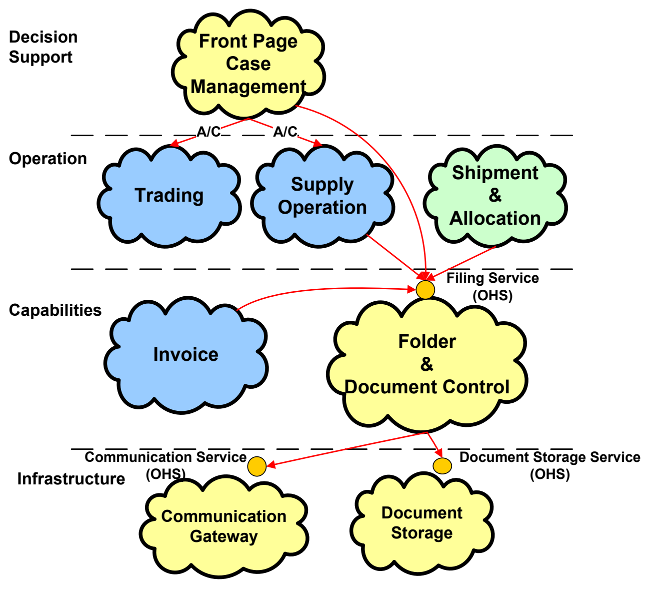

Figure 4. Responsibility layers illustrated

5. Conclusion

The experience from our use of strategic level Domain-Driven design is that context maps and the activity of context mapping can improve the quality of the Enterprise Architecture and its derived software architectures as well.

Another finding is that being able to identify the projects core domain is important with respect to how to utilize development resources, and how developers chooses to think about the software under development. As an example, accepting that the core domain in Digital Cargo Files is found in the Front Page / Case Management context, and not in the document storage model increases the developers understanding of the software and its future use.

The encountered challenge is for the business to agree on what is most important, where one of the discoveries is that in large scale systems such as the WSC there will be multiple cores.

The third finding was that the combination of context maps and responsibility layers reduces the perceived complexity.

In summary our experience is that strategic level Domain-Driven design can be used to enhance Enterprise Architectures and the derived software architectures.

5.1 Further work

It is our perception that the use of context maps and their role in architectural improvement is well understood. As an example the DCF system will be refactored in the fall 2006 to match the suggested recommendations found in this paper. Statoil has also formally adopted use of context-maps as an architectural artefact.

When it comes to use of distillation and large scale structures and their potential role in Enterprise Architecture and derived software architectures are not truly understood, but use of responsibility layers seems to reduce the perceived complexity [5].

In the case of distillation it might be argued that the technique is more tactical than strategic. Shipment & Allocation (Figure 4) represented a 20.000 hour development effort. Knowing which parts of that large chunk of software is its core is important for the development project as it should focus its effort on those parts. A discussion whether Shipment & Allocation is part of the core of the WSC or not, feels meaningless.

With respect distillation and large scale structures we only have scratched the top of the iceberg, and we invite other researchers and practitioners to participate in further research.

6. Acknowledgments

Our thanks go to: Eric Evans for his insights and contributions with respect to the content of this paper. We also would like to thank him for being a skillful mentor and coach. In addition, we would also like to thank Olaf Zimmermann and the ACM rehearsal staff for their contributions, and at last but not least, our colleges at Statoil for all the constructive and demanding discussions.

7. References

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

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

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

[4] Domain-Driven design, http://domaindrivendesign.org.

[5] Hitchins D. K. Advanced Systems, thinking, engineering and management, Artech House, 2003, ISBN 1-58053-619-0.

[6] Zachman J., http://www.zifa.com.

[7] Rechtin E. and Maier M. The art of systems architecting, CRC Press, 2002, ISBN 0-8493-0440-7.

[8] Service Oriented Architecture (SOA), http://en.wikipedia.org/wiki/Service_oriented_architecture.

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

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

[11] Wesenberg, H., Landre, E., and Rønneberg, H. Using Domain-Driven Design to evaluate commercial-off-the-shelf software, OOPSLA 2006.