Seminar Design Patterns und Anti-Patterns

Architecture Anti-Patterns

Arkadiusz Gabryś
Matrikel-Nr. 21895081
Department of Compuer Science

July 22, 2015

Abstract — Content of this document describes three common Architectural Anti-Patterns. Sources and causes, symptoms and consequences, examples and solutions will be discussed.

Contents

1 Introduction

1.1 Software Architecture

Good architecture is a critical factor in process of transforming high level models into actual implementation [3]. Unfortunately the engineering discipline of software architecture is relatively immature and there is even no official definition [4]. SEI provides three types of software architecture definitions [5]:

- Modern definition – The set of structures needed to reason about the system, which comprises software elements, relations among them, and properties of both.
- Classic definitions – An architecture is the set of significant decisions about the organization of a software system, the selection of the structural elements and their interfaces by which the system is composed, together with their behavior as specified in the collaborations among those elements, the composition of these structural and behavioral elements into progressively larger subsystems, and the architectural style that guides this organization.
- Bibliographic definitions – Software architecture is the study of the large-scale structure and performance of software systems. Important aspects of a system's architecture include the division of functions among system modules, the means of communication between modules, and the representation of shared information.

The common aspect of those definitions is the placement of software architecture as a layer between high level abstract model and specific implementation.

1.2 Architecture Anti-Patterns

Anti-Pattern is a commonly used solution that results in bad consequences It describes the practice itself as well as solution [6]. Architecture Anti-Patterns are those patterns that occur at architecture level, so called system-level or enterprise-level [1].

In this document, while describing anti-pattern, firstly the general form of a bad solution will be presented both with it causes. Next the main symptoms and consequences will be presented and after that some real life example to describe example solution. In case of anti-patterns there occur some exceptions - situations when an anti-pattern can be accepted - and that will be the last chapter for each one of them.

This whole project (excluding graphics) was created using LaTeX technology and its sources are available at https://github.com/gabr/dpap_aap

2 Stovepipe Enterprise

2.1 Form & Causes

Stovepipe Enterprise, also know as Island of Automation [8] is a situation when multiple system within enterprise are designed independently at every level [10]. Those systems has potential to share data, functionality or whole subsystems but they don't.

Stovepipe metaphor isn't accidental. Stovepipe is the pipe which conducts smoke from a coal or wood-burning stove to it's chimney. There are two problems with such pipes and two similar problems will occur in case of stovepipe enterprise anti-pattern. The first problem is that burning wood produces corrosive substances that erode metal, so pipe must be constantly maintained and repaired in order to avoid leakage. Second problem is fact that stovepipes are never connected with each other to create one system. If one would have a two coals he would have two separate stovepipes to maintain.

This analogy perfectly fits into stovepipe enterprise anti-pattern where we have separated systems which has to be maintained separately because of they layer separation. This and other problems will be discussed in the next chapter.

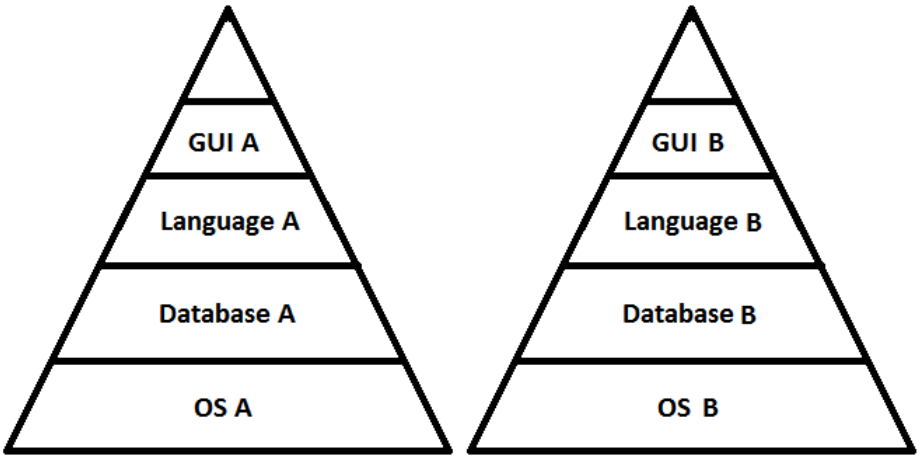


Figure 1: Stovepipe Enterprise Systems - example visualization

Causes of such situations are in general lac of standards, lac of communication and laziness. When a company have global system standards, technology strategy and system profiles then even without teems and enterprise institutions communication such anti-pattern could be avoided. Lack of communication, lack of knowledge about technology standards being used in company and absence of horizontal interfaces in system integration solutions are the main causes of the stovepipe enterprise anti-pattern [1]. It can also came from rapid expansion of the company and systems - from few small applications to big enterprise systems [2].

2.2 Symptoms & Consequences

The main symptoms and consequences of the stovepipe enterprise anti-pattern are [1]:

- Lack of software reuse between enterprise systems
- Lack of interoperability between enterprise systems
- Brittle systems
- Monolithic systems

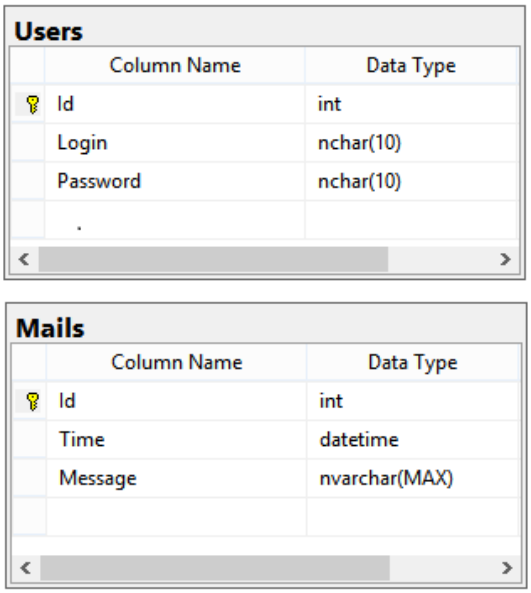
- Undocumented architectures
- Inability to extend systems
- Excessive maintenance costs
- Employee turnover may causes project discontinuity and maintenance problems

Many of those consequences can be notices at the early stages of system creation. If some subsystem can be maintained only by one employ this is probably separate "island" of the system. If system architecture is known only to system team and there is no enterprise standard then undocumented architecture problem will occur. Also no one else outside the team would be able to quickly join to the team or connect two different systems.

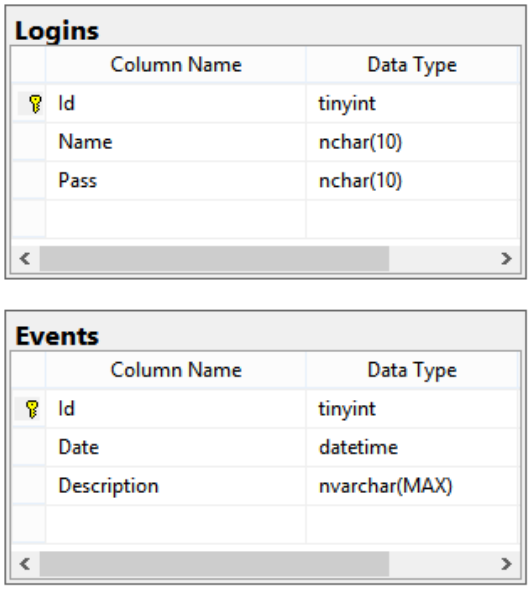
2.3 Example - authorization system

As an example of the stovepipe enterprise anti-pattern will be considered a e-mail system and calendar system within one company.

Both systems besides they unique data had to store information about they users and passwords. In enterprise where are no standards on database layer defined we can expect that those systems can use different technologies, naming conventions and will probably not share the information. Figure 2 shows tables for those two systems. In case where those systems uses different databases technologies the gap between database layer is even higher.



(a) E-mail system - example tables in MySQL database



(b) Calendar system - example tables in MSSSL database

Figure 2: Stovepipe Enterprise Systems database example - different tables in different databases technologies

Not only the the separation of tables is the symptom of the stovepipe enterprise anti-pattern but also the different naming conventions and *Id* field data type. Such systems require double work in case of changing any user data in any of those two systems. So it is inconvenient both in development and in administration.

2.4 Solution*

[1] Coordination of technologies at each level is necessary to both avoid and solve stovepipe enterprise problem. Providing any enterprise design pattern will help future refactorization. Defining standards at each level is the first step and it should be done in order from the lowest level (high-level architecture) to the highest (API specifications, extensions etc).

Topic is new but the problem is old, so many large enterprises developed conventions for the definitions of object-oriented architectures that can be applied to many organizations. The key is to define detailed interoperability [2] conventions across systems of large-scale architectures. And at the same time address technology strategy and requirements. Experience has shown that four requirements models and four specification models has to be defined in order to properly scope interoperability between layer.

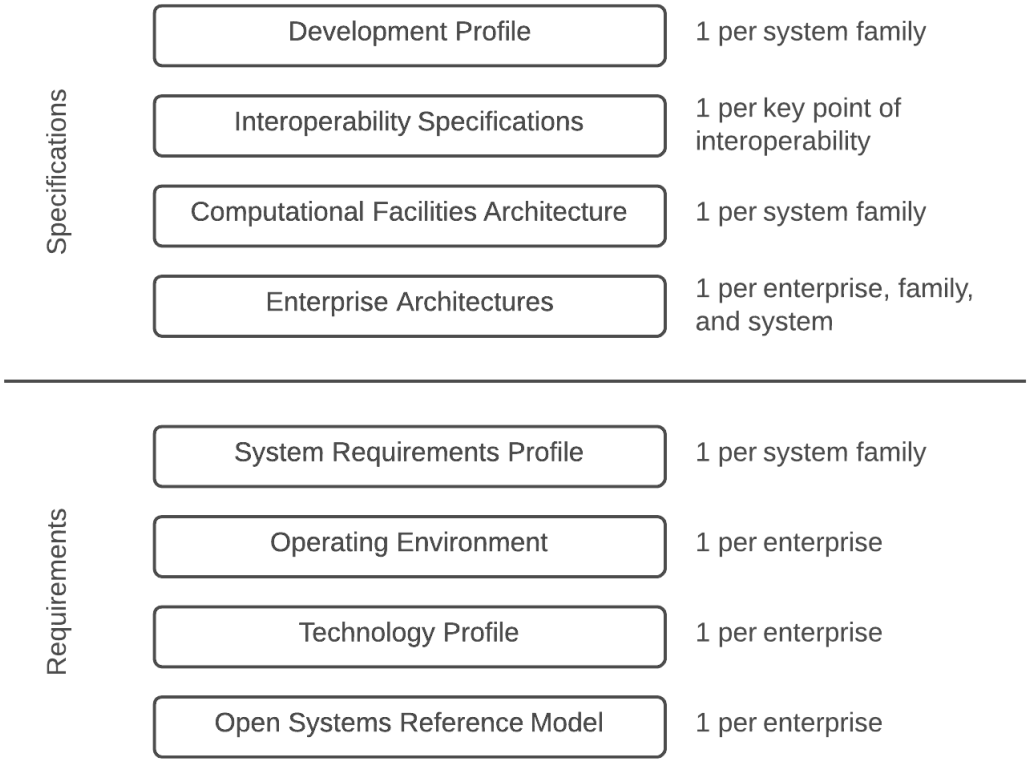


Figure 3: Scoping of interoperability [1]

Detailed description of each layer will not be discussed.

[1] Due to lack of other source the whole chapter is based on [1]

[2] Property of a product or system, whose interfaces are completely understood, to work with other products or systems, present or future, without any restricted access or implementation. [7]

Based on that, the solution for the example database layer show on figure 2 can be provided. The first step will be to define one database technology, then the naming conventions and types standards. After that one table for all user for both systems can be defined as it is shown on figure 4.

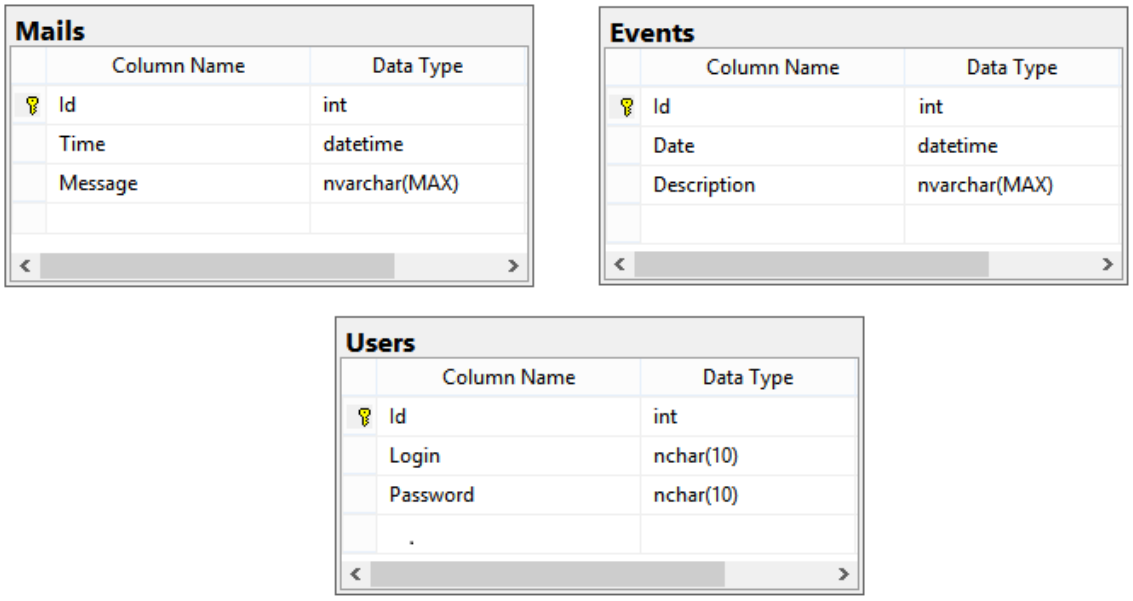


Figure 4: Example database layer - solution

2.5 Exceptions

Stovepipe Enterprise Anti-Pattern may occur in situations in which is acceptable. In new systems and application is not, however when companies grow by takeover or merge occurrence of this anti-pattern can be expected. Vendor Lock-In Anti-Pattern can be an other exception [1]. The situation when company grows by itself, when it starts from few small applications which are not well designed, there will be a point problems connected to stovepipe enterprise will show up [2].

Besides Vendor Lock-In exception, all others are just temporary and should be refactored at the earliest state as possible.

3 Stovepipe System

3.1 Form & Causes

Stovepipe System Anti-Pattern occur within one system. Two definitions can be found:

- Term describes system developed to solve a specific problem and containing data that cannot be easily shared with other systems [8].
- System where no abstraction layers are used to communicate between subsystem which results in situation where everything is connected with everything [11, 12].

The first one is more general and applies to all stovepipe anti-patterns. The second one will be further discussed in next chapters.

It happens in systems which integrates different subsystems to achieve they goal. But direct causes of this anit-pattern is no architectural vision which leads to lack of already mentioned abstraction in the system providing tight coupling between implemented subsystems [1].

3.2 Distinction

When describing stovepipe anti-patterns the clear distinction between them should be provided. Example difference are listed in figure 5.

Stovepipe Enterprise	Stovepipe System
Multiple systems	One system
Lack of common layer between systems	Lack of common abstract layer between subsystems
Caused by:	Caused by:
- Lack of communication	- Lack of architectural vision
- Lack of standards	- Tight coupling between classes

Figure 5: Distinction between Stovepipe Enterprise and Stovepipe System

3.3 Symptoms & Consequences

One of the main symptoms is difficulty in both modifying the system and describing its architecture. This leads to large semantic gap between architecture documentation and implementation and software will probably not meet the user expectations even when code is comply with the paper requirements [11]. System maintenance will become increasingly costly with both time and money [1, 11, 12]. Developers will have to invent workarounds and most attempts to automate system will fail [1].

3.4 Example

Example stovepipe system shown on figure 6 can be divided into four types of components:

- Clients
- Scanners
- Printers
- Databases

In the example all subsystems are connected together and there is no abstraction layers between components. Each client has its own code to manage all devices and other clients.

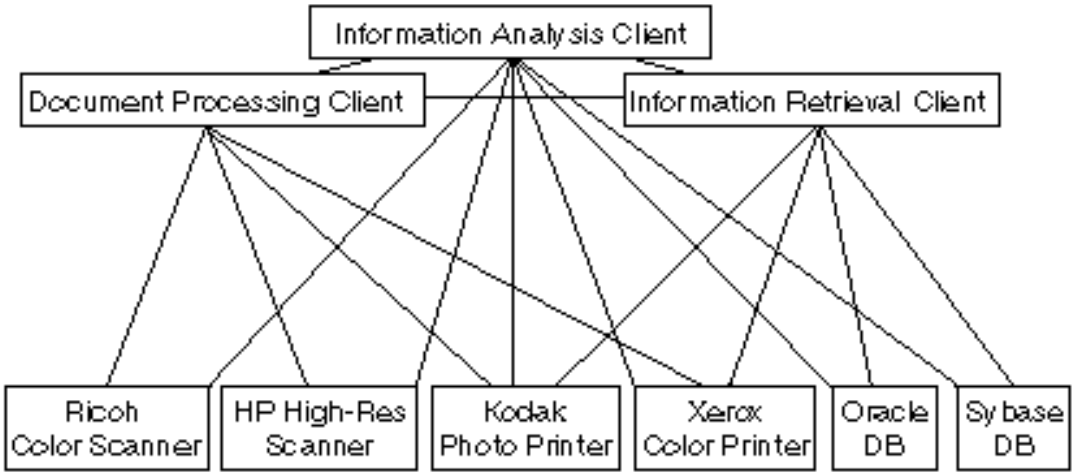


Figure 6: Stovepipe System - example diagram [12]

3.5 Solution

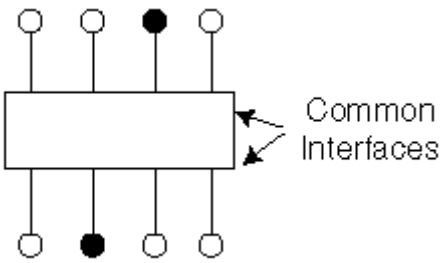


Figure 7: Interface between subsystems [12]

Refactoring of stovepipe system begins from extracting abstract models for subsystems. Abstract models will allow reuse of code and adding new subsystem will not require code changes in any of clients. Discovering the appropriate abstraction is the key to defining component interfaces. This will model subsystems interoperability without exposing unnecessary differences between subsystems [12].

Summarizing: non of the subsystems should communicate without abstraction layer.

For our example proper abstraction architecture may look like it is shown on figure 8. From base graph shown on figure 6 three types of subsystems interfaces can be extracted along with client interface - because each of the client is also connected with other clients. Those types are:

- Input devices
- Output devices
- Information sources

and client interface.

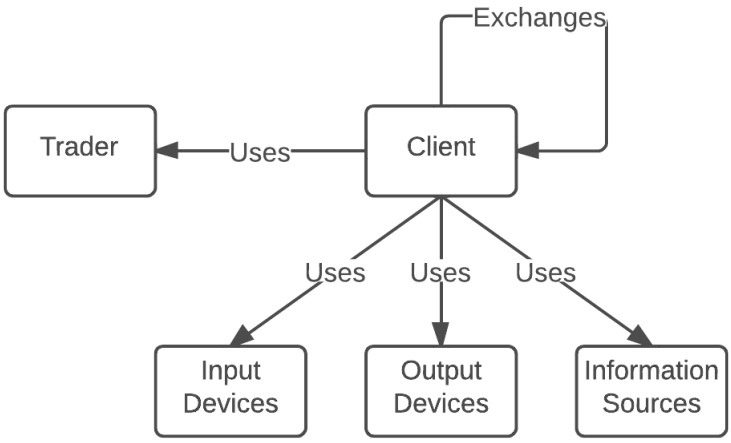


Figure 8: Stovepipe System - abstraction to example diagram [12]

There is now direct communication and no tight coupling between components which allows to easily add or remove subsystems and clients.

3.6 Exceptions

There are situations which justify this anit-pattern in project. One of those is exploration of new or unknown domain - creating prototypes. Also quick development may end up as stovepipe system but this is acceptable for achieving rapid solutions [1, 11].

4 Vendor Lock-In

4.1 Form & Causes

In general it is strong dependence on a particular vendor technology which makes difficult or expensive to switch to an other technology [13]. This especially problem when using vendors technology which is not based on global or open standards [9]. Each vendor update or change make problems to continuously maintain system interoperability [1].

One of causes of this anti-patter is selecting product basing on marketing or sales information instead of technology inspection. Selected product varies from published open system standards because there is no conformance process which leads need for deep in knowledge about the product. Also thigh coupling with vendor technology is always bad practice - avoiding of developing isolation layer [1].

4.2 Symptoms & Consequences

Depending upon vendor technology will drive application maintenance cycle. This can evolve to the problem when promised product features are delayed or never delivered because of vendor failures. In other hand missing product update will often end up on complete reintegration from the beginning [1]. Attempts to change vendor technology is costly, difficult or impossible because of market monopoly.

4.3 Example

Typical example on software level are file formats. File formats that are not easily convertible to other formats will force they users to stick with one software that is able to use it [13]. Real world example will be Microsoft Word, Outlook and Excel [14].

4.4 Solution*

[3] To avoid problems with changing vendor technology or to deal with vendor software updates problem and isolation layer should be provided. It should be used in cases when vendor technology is low level one or there is need to provided more convenient programming interface is necessary.

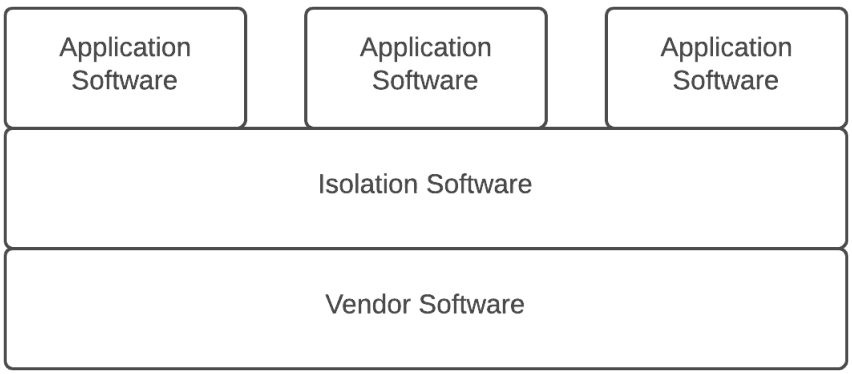


Figure 9: Vendor Lock-In Isolation Layer [1]

This layer can and should provided following functionalities:

[3] Due to lack of other source the whole chapter is based on [1]

- Handling default behaviors
- Errors handling
- Cross platform/system interface
- Version inspection
- Types and formats conversions

Benefits from this solution are such that migration of the system to an other vendor will not require changes in systems but only in isolation layer. This will lower costs and risks of such operation. There is also separation of infrastructure, low level knowledge from application knowledge which allows one team to maintain isolation layer and all others to develop applications without facing all problems connected with vendor product.

But isolation layer has to be maintained and coordinated. Requires additional development effort and can produce additional problems. While designing developers must be coordinated to avoid stovepipe enterprise anti-pattern.

4.5 Exceptions

This anit-pattern can be acceptable when a single vendor's code makes up the majority of code needed in an application [1]. Also when the whole enterprise is using the same (but not low level) technology provided by single strong company it will many not cause problems related to this vendor lock-in anti-pattern.

5 Summary

Architectural Anti-Patterns may not be a problem in small projects or companies. But they are a real boundary in reaching high performances in big enterprises and systems. Interoperability is crucial for technology and companies cooperation which benefits all.

While stovepipe anti-patters are just a result of bad design, the vendor lock-in is often a intentional practice to keep customers or gain more from product in the future. So by some companies it may be seen as marketing design pattern and many of the leading enterprises are using it.

They are not easy to solve inconvenientes, requiring a lot of effort to prevent them. Knowledge about the problem at the early state of development is the key to minimize all costs. Because those are not new problems there already exists solutions which may be adapted by anyone.

List of Figures

References

[1] Alexander Shvets *AntiPatterns The Survival Guide*, published by https://sourcemaking.com

[2] W.H. Immon *Slaying the stovepipe dragon* 24 April 2003 p. 2

[3] *Nordic Workshop on Model Driven Engineering* ISBN: 978-91-7295-985-9, 27-29 August 2007 Ronneby, Sweden p. 5

[4] Wikipedia *Architektura oprogramowania*, web: https://pl.wikipedia.org/wiki/Architektura_oprogramowania 10 July 2015

[5] Software Engineering Institute, Carnegie Mellon University *What is your definition of software architecture?*, web: http://www.sei.cmu.edu/architecture/start/glossary/definition-form.cfm 10 July 2015

[6] Brad Appleton *Patterns and Software: Essential Concepts and Terminology* 19 September 2008 12:06 AM p. 7

[7] AFUL dedicated website for Definition of Interoperability http://interoperability-definition.info/ 20 July 2015

[8] *Stovepipe Anti Pattern* web: http://c2.com/cgi/wiki?StovepipeAntiPattern 11 July 2015

[9] *Vendor Lock In* web: http://c2.com/cgi/wiki?VendorLockIn 11 July 2015

[10] Brown, W., Malveau, R., McCormick *AntiPatterns Refactoring Software, Architectures, and Projects in Crisis* III, H., Mowbray, T., John Wiley & Sons, Inc., 1998

[11] Christian Schallhart *Transaction Processing for Clustered Virtual Environments* Institut für Informatik Technische Universität München p. 8

[12] Hays W. McCormick, Raphael Malveaux *Solutions for refactoring software* Dr. Dobb's Journal June 1998 web: http://collaboration.cmc.ec.gc.ca/science/rpn/biblio/ddj/Website/articles/DDJ/1998/9806/9806d/9806d.htm#rf5

[13] *Vendor Lock-in Definition* web: http://www.linfo.org/vendor_lockin.html

[14] Wikipedia *Vendor lock-in* web: https://en.wikipedia.org/wiki/Vendor_lock-in