Comparing the Maintainability of two Alternative Architectures of a Postal System: SOA vs. non-SOA

Maurizio Leotta, Filippo Ricca, Gianna Reggio, Egidio Astesiano

Abstract:

Recently, we were prompted by a local company to improve the maintainability of a postal legacy system and reduce the time to close the change requests.

In this paper, we describe the first step of our on-going project that consists in the comparison of two alternative architectures of the target postal system using the Software Architecture Analysis Method (SAAM). The first is the architecture currently used in the postal system while the second one is a new architecture based on SOA. Preliminary results are in favour of the new architecture that can be adapted to change faster than the actual one.

The main lesson that we have learnt in this project is that SAAM is a methodology simple to apply and useful for comparison purposes but the effective final result remains somewhat subjective.

Digital Object Identifier (DOI):

`http://dx.doi.org/10.1109/CSMR.2011.41`

Comparing the Maintainability of two Alternative Architectures of a Postal System: SOA vs. non-SOA

Maurizio Leotta, Filippo Ricca, Gianna Reggio, Egidio Astesiano
Dipartimento di Informatica e Scienze dell'Informazione - DISI
Università di Genova, Italy
Email: maurizio.leotta|filippo.ricca|gianna.reggio|astes@disi.unige.it

Abstract—**Recently, we were prompted by a local company to improve the maintainability of a postal legacy system and reduce the time to close the change requests.**

In this paper, we describe the first step of our on-going project that consists in the comparison of two alternative architectures of the target postal system using the Software Architecture Analysis Method (SAAM). The first is the architecture currently used in the postal system while the second one is a new architecture based on SOA. Preliminary results are in favour of the new architecture that can be adapted to change faster than the actual one.

The main lesson that we have learnt in this project is that SAAM is a methodology simple to apply and useful for comparison purposes but the effective final result remains somewhat subjective.

Keywords-**Architecture Analysis, Architectures Comparison, Re-engineering, Legacy postal system, SAAM, SOA**

I. Introduction

The authors have been recently involved by a local company in the restructuring/re-engineering of a postal system. The system, named XYZ[1], is a large non object oriented legacy system (more than 1 million lines of code, 2000 files and 200 executable modules) written in several languages (i.e., C, C++, VB, C#, VBSCRIPT and PL/SQL) and running in a distributed environment. Acquisition and processing are executed on the central site, while the production takes place in one of the peripheral sites (called printing centers). XYZ is used by postal organizations that offer their customers, such as big companies (e.g., banks), specific services in order to manage big amounts of physical mail (e.g., invoices and bank balances) starting from electronic data files given in several formats (e.g., XML and PDF).

The restructuring/re-engineering of the postal system is motivated by the need of having a more agile IT system that can be adapted to change faster. In the last ten years, emerging technologies (e.g., new printers and enveloping machines), new kind of inputs and outputs (e.g., new document formats), new laws/rules disrupting the current requirements and new users' needs have changed and expanded the functionalities required to a postal system.

[1]For privacy reason, we cannot report here the name of the system.

The company is mainly interested to understand whether: (i) the long cycles of code maintenance needed to satisfy the customers' requests are mainly due to the *Actual Architecture*, which is degraded over time and (ii) a *New Architecture* of the System based on SOA could improve the current situation. Thus, the first step of the project consists in analyzing the *Actual Architecture* and understanding whether a *New Architecture* based on SOA [2] could reduce the maintenance/evolution times.

The first problem that we faced was deciding a strategy and a usable methodology to carry out the comparison. After several discussions and the search of a methodology for the comparison, we decided to proceed as follow. We have first analyzed and reverse engineered XYZ, helped by the experts of the system and by the documentation, obtaining the *Actual Architecture* description. Then, we sketched a *New Architecture* description, based on modern paradigms, paying attention particularly to maintainability and evolvability. Finally, we compared the two architectures using the Software Architecture Analysis Method (SAAM) [1], [3]. SAAM is a method for predicting system level quality attributes (in our case maintainability) based upon software architectural evaluation. SAAM makes use of the concept of scenario [1] in order to evaluate competing architectures.

The work is still ongoing and here we only report some preliminary results of the comparison.

The paper is organized as follows. Section 2 presents the key ideas of SAAM. Section 3 describes the competing architectures. Section 4 reports the comparison and discusses the preliminary results. Finally, Section 5 reports the lessons learnt and Section 6 concludes the paper sketching some future work.

II. SAAM

SAAM [1], [3] consists of four steps: describing architectures, developing scenarios, performing scenario evaluations and performing overall evaluation. In the following, we describe how we have instantiated those steps in our case.

1) **Describing Architectures**. We have produced a high-level description of the two architectures using a simple syntactic architectural notation. The *Actual*

Architecture description has been created by analyzing the documents provided by the company, reverse engineering the code and consulting the experts of the system. To reach an agreement on the *Actual Architecture*, several meetings with the experts of the system were needed. Instead, the *New Architecture* description has been developed by ourselves following classical key ideas for a good design (e.g., high cohesion and low coupling among components) and paying attention particularly to maintainability and evolvability. In that last task, we were inspired by object oriented and SOA principles.

2) **Developing Scenarios**. A scenario is a brief description (typically one sentence long) of some anticipated or desired use of a system [1]. SAAM proposes to use scenarios as a tool for the analysis of the quality of a system. Scenarios express the particular instances of each quality attribute important for the customer of a system. The architecture under consideration is analyzed with respect to how well or how easily it satisfies the constraints imposed by each scenario [1]. In our case, scenarios are change requests (CRs) and the considered quality attribute is maintainability/evolvability of the system. We used SAAM to show which is the best architecture (i.e., the simpler to modify) with respect to the considered CRs. For this reason, we and the company have created a list of scenarios that capture relevant and real change requests emerged by the customer's company needs.
3) **Performing Scenarios Evaluation**. In this step we have analyzed the impact of each scenario on the two architectures. In detail, for each scenario we have described which components of the *Actual/New Architectures* need to be modified to fit the scenario and if there is the necessity to add new components to the two architectures. Finally, for each modification or addition we have estimated the accomplished effort. In this way, it is possible to compare the two architectures for each considered scenario in terms of the number of components to modify/add, and in terms of the effort to be done.
4) **Performing Overall Evaluation**. In this step we have summed up the results of each scenario evaluation and grouped them in a summarizing table. Thus, we can assess which architecture require less modifications/additions and less effort to fit all the considered scenarios. In this way it is possible to give to the company a preliminary indication of the best architecture with respect to the considered CRs.

III. Architectures

In this section we briefly describe the two high level alternative architectures for XYZ. We represented them using a simple syntactic architectural notation as suggested in [3]. We agree with the authors of [3] that: "*The candidate architectures should be described in a syntactic architectural notation that is well-understood by the parties involved in the analysis*." These architectural descriptions indicate the systems computation, data components and all component relationships (i.e., the connectors). The rectangles represent the computational components of the System while the arrows represent the control/data flow between different components. Finally, the cylinders represent databases.

The *Actual Architecture*, that reflects the real architecture of the current System, is sketched in Figure 1.

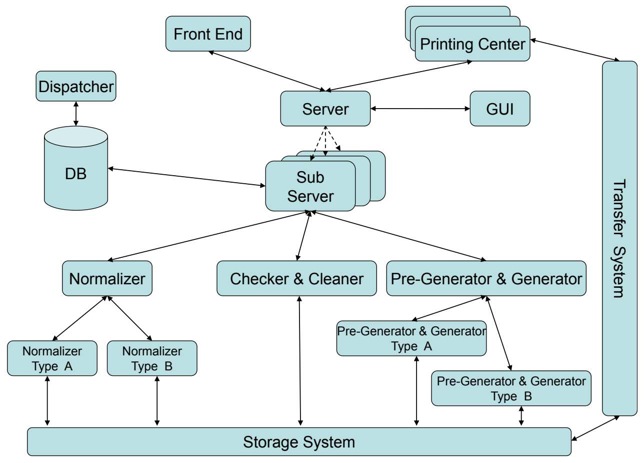

Figure 1. *Actual Architecture* description

Electronic files (batches) submitted by the customers, are acquired by the `Front End` that sends them to the `Server`. The Server handles in parallel all the received files: batches are normalized, checked, cleaned, pre-generated, dispatched, generated and, in the end, sent to the peripheral `Printing Centers`. Printing Centers carry out the following phases: printing on paper, folding, enveloping and boxing. For each received batch, the Server create a `SubServer` process that, using the components `Normalizer`, `Checker & Cleaner` and `Pre-Generator & Generator`, transform that batch inch by inch. To execute their task, these last components call more specialized components (e.g., `Normalizer Type A`). For each type of format managed by the system, there exists a component that specializes `Normalizer` and `Pre-Generator & Generator`. All the relevant information is stored in the database `DB`, which also contains a set of Stored Procedures (`Dispatcher`) used to perform splitting and delivery of the batches to the `Printing Centers`. The files generated during the entire transformational process are stored in the `Storage System`. Instead, the task of the `Transfer System` is to transfer the elaborated batches to the Printing Centers. Finally, the human operator (not shown in Figure 1) can set several parameters of the System by means of a `GUI`.

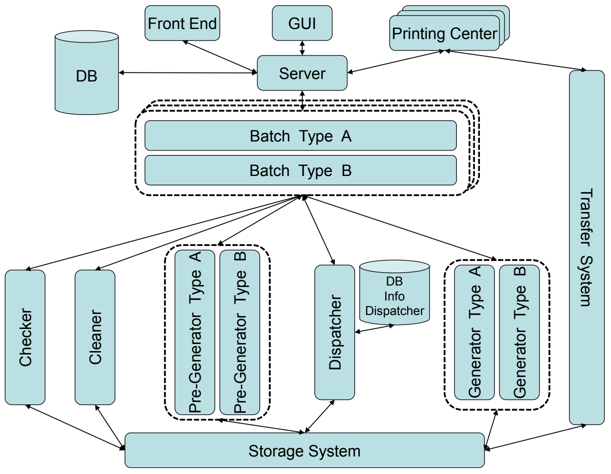

Figure 2. *New Architecture* description

Figure 2 describes the *New Architecture* for the System. In this architecture several components have not been substantially changed (i.e., GUI, Front End, DB, Printing Centers, Transfer System and Storage System), while others have been significantly modified. The core of the system, i.e., the portion of the system that oversees the management of the batches, has been completely re-engineered following the SOA paradigm[2]. Now, the `Server` manages the parallel execution of several running orchestrators (the `Batches`). The orchestrators call the Web services `Checker`, `Cleaner`, `Pre-Generator`, `Dispatcher` and `Generator` able to fulfil the various tasks. Note that in the *New Architecture*: (i) the splitting of the batches, realized in the *Actual Architecture* by a set of Stored Procedures, is realized by the Web service `Dispatcher`. (ii) the `Checker & Cleaner` and `Pre-Generator & Generator` of the *Actual Architecture* are each subdivided in two Web services. (iii) the logic of the `Normalizer` is partially included in the `Server` and simplified due to modifications of the input.

The *New Architecture* has been built by instantiating the Multi-Translation Frame presented in [4]. Moreover, to show its feasibility, we have implemented in Java a simplified version of the postal system with the *New Architecture*, producing a working prototype.

IV. Evaluations between Actual and New Architectures

In this section the remaining three steps of SAAM are applied to the two architectures presented above.

[2]Really, at this level of abstraction SOA or OO make no difference. Each component of the core of the system, can be indifferently seen as a Web service or a OO component, i.e., as a communicating computational entity providing a set of operations/capabilities.

A. Scenarios Description

A fundamental step of SAAM is the scenarios development; the comparison between the two architectures is conducted considering each scenario produced in this phase. With the help of the experts of the system, we have created a list of ten scenarios able to capture relevant and real change requests emerged by the customer's company needs.

An example of scenario considered in this work is `new_input_format` that consists in: "*modifying the postal system so that it is able to accept and handle a new input batch format called C*". As we have seen, the *Actual Architecture* can receive inputs in a fixed number of formats. That change request aims to extend the set of managed formats with the new format C. The `new_input_format` CR will impact several components: the components accepting and using the inputs but also the components that work with and transform the data submitted by the customers (e.g., `Normalizer` in the *Actual Architecture*).

B. Performing Scenarios Evaluation

In this phase, we have analyzed the impact of each scenario on the two architectures. In detail, for each scenario, we have: described which components of the *Actual* System and which components of the *New* System require changes to fit the scenario (modification) and if there is the necessity to add new components to the Systems (addition). Finally, for each introduced modification/addition we have estimated the effort to be done (i.e., low, medium, high). In this way it is possible to determinate which architecture better fit the scenario in terms of number of modified/added components, and in terms of accomplished effort.

As an example, we show the impact of the `new_input_format` CR over the two architectures. Table I describes which changes are needed to implement `new_input_format` in the *Actual* System, while Table II shows the impact of the `new_input_format` CR on the *New* System.

Table I
IMPACT OF `NEW_INPUT_FORMAT` CR ON THE *Actual Architecture*

Component	Description	Type[a]	Effort
Normalizer	management of the new Normalizer Type C	Mod	Light
Normalizer Type C	management of the tool for the format C	Add	Medium
Pre-Generator & Generator	management of the new Pre-Generator & Generator Type C	Mod	Medium
Pre-Generator & Generator Type C	management of the tool for the format C	Add	Medium
SubServer	introduction of a new type of flow	Mod	Light

[a]Mod means Modification and Add means Addition.

Looking at the two Tables, it is apparent that there is a little advantage for the *New Architecture*. The number of

Table II
IMPACT OF NEW_INPUT_FORMAT CR ON THE *New Architecture*

Component	**Description**	**Type**	**Effort**
Server	management of a new type of Batch / Input	Mod	Medium
Batch Type C	introduction of a new type of Batch	Add	Light
Pre-Generator Type C	introduction of a new type of Pre-Generator	Add	Light
Generator Type C	introduction of a new type of Generator	Add	Light

impacted components is 5 for the *Actual Architecture* and only 4 for the *New Architecture*. Moreover, the effort is greater in the case of the *Actual Architecture* (3 medium and 2 low effort operations for the *Actual Architecture* vs. 3 low and 1 medium effort operations for the *New Architecture*).

C. Performing Overall Evaluation

The last step of SAAM is executing an overall evaluation of all the considered scenarios on the two architectures with the aim of assessing which is the best one. It means grouping the results of each scenario evaluation in a summarizing table. To execute the comparison, we considered both the number of components modified/added[3] and the estimated effort. Clearly, the best architecture is that requiring less additions/modifications and less effort. Table III clearly shows that the *New Architecture* requires less changes (modifications plus additions) to the various components to accommodate the considered ten scenarios. Globally, the *Actual Architecture* needs 35 changes while the new one only 27. Furthermore, the *New Architecture* also requires a lower total effort.

Table III
TOTAL SCENARIOS IMPACT COMPARISON

Arch	**Component Impacted**	**Type**		**Effort**		
		Add	**Mod**	**Low**	**Med**	**High**
Actual	35	30	5	21	11	3
New	27	19	8	17	8	2

In conclusion, we can state that the *New Architecture* can be adapted to change faster than the *Actual Architecture*.

It is important to note that this outcome strongly depends on the ten scenarios chosen. However, this is a significant achievement because the ten scenarios have been supplied by the company and they are real change requests reflecting the needs of the customers.

V. LESSON LEARNT

Comparing two architectures for a system with the aim of understand which is the best with respect to a criterion is not an easy task. That task is made more difficult, as in our case, if one architecture is well known whereas the other one is hypothetical, more immature and created only to answer to a research question. In such case, the main difficulties are understanding whether: (i) the evaluation of the impact on the *New Architecture* is realistic and (ii) the *Actual Architecture* does not lose the comparison only because is easier finding and computing the impact in a well-known architecture than in a hypothetical one.

In this preliminary work we have experimented that SAAM is a methodology very useful for organizing a process of architectural comparison. Altogether, it is simple to apply even if the effective final result remains somewhat subjective. In several cases, also with the constant support of the experts of the postal system, we had some problems to identify the impact of the CRs on the two architectures and to presuppose the effort. Another problem that we have observed is related to the abstraction level of the two architectures. It is difficult to sketch two architectures at the same level of abstraction, especially if, as in our case, they follow two different paradigms.

VI. CONCLUSION AND FUTURE WORK

In this paper, we have described the first step of our on-going re-engineering/restructuring project. It mainly consists in the comparison of two alternative architectures of a legacy postal system using SAAM. Preliminary results are slightly in favour of the *New Architecture*: it can be adapted to change faster than the *Actual Architecture*.

Future works will be devoted to extend the number of CRs to have a more reliable comparison and to complete the first step of the project. The Company will decide how and if to proceed with the next steps of the project depending on the conclusive result of the comparison. In particular, the next step will be, probably, a migration towards SOA (re-engineering) if the comparison will be completely in favour of the *New Architecture*. Otherwise, it will simply consist in a restructuring step at level of code (e.g., using static analysis tools to remove clones and code smell detectors to identify opportunities to improve the postal system through refactoring).

[3]We decided to equally weight modification and addition even if usually modifications require more effort. We opted for this solution because it is difficult to set a reliable proportion different from 1.

REFERENCES

[1] P. Clements, L. Bass, R. Kazman, and G. Abowd. Predicting software quality by architecture-level evaluation. In *Proc. of International Conference on Software Quality*, volume 5, pages 485–497. Software Engineering Institute, 1995.

[2] T. Erl. *SOA Principles of Service Design*. Prentice Hall, 2007.

[3] R. Kazman, G. Abowd, L. Bass, and P. Clements. Scenario-based analysis of software architecture. *IEEE Softw.*, 13(6):47–55, 1996.

[4] G. Reggio, E. Astesiano, F. Ricca, and M. Leotta. A problem frame-based approach to evolvability: the case of the multi-translation. In *16th Monterey Workshop*, Microsoft Center, Redmond, US, 2010. (Submitted).