Migrating a Large Scale Legacy Application to SOA: Challenges and Lessons Learned

Ravi Khadka*, Amir Saeidi*, Slinger Jansen*, Jurriaan Hage*, Geer P. Haas†
*Department of Information and Computing Sciences, Utrecht University
{r.khadka, a.m.saeidi, slinger.jansen, j.hage}@uu.nl
†IBM, The Netherlands
geer.haas@nl.ibm.com

***Abstract*—This paper presents the findings of a case study of a large scale legacy to service-oriented architecture migration process in the payments domain of a Dutch bank. The paper presents the business drivers that initiated the migration, and describes a 4-phase migration process. For each phase, the paper details benefits of using the techniques, best practices that contribute to the success, and possible challenges that are faced during migration. Based on these observations, the findings are discussed as lessons learned, including the implications of using reverse engineering techniques to facilitate the migration process, adopting a pragmatic migration realization approach, emphasizing the organizational and business perspectives, and harvesting knowledge of the system throughout the system's life cycle.**

I. Introduction

In the current business environment, enterprises are pressured to respond to changes in the market, laws and regulations, and to remain efficient and innovative to reap benefits from on-demand and new business opportunities. In order to manage these changes and remain competitive, flexibility is required within the enterprise, supported by technology [1]. Technology support itself is constantly evolving with the advancement of new computing paradigms and improvements in hardware infrastructures. Enterprise systems should therefore be designed to enable continuous evolution and to remain responsive to new business opportunities, realizing better reuse and maintainability, and to improve business-IT alignment to achieve business goals [1]. One of the obstacles to adapt to such changes is the presence of legacy systems [2]. Despite their well-known disadvantages, such as being inflexible and hard to maintain, legacy systems are still vitally important to enterprises as they support complex core business processes; they cannot simply be removed as they implement and execute critical business logic effectively and accurately. Unsurprisingly, the knowledge contained in these systems is of significant value to an enterprise. On the other hand, proper documentation, skilled manpower, and resources to evolve these legacy systems are scarce. Therefore, momentum is growing to evolve those legacy systems within new technological environments such as Service-Oriented Architecture (SOA) as SOA facilitates the reuse of existing assets [3]. The SOA paradigm is favored by loose-coupling, flexible composition of business services, re-usability, and abstraction from the underlying technology platforms. Hence, migration from legacy systems to SOA enables enterprises to achieve flexibility for collaboration, agility within a constantly changing environment [3] and thus enabling business-IT alignment. With these claimed benefits, there has been an increasing interest in academia to investigate approaches for migrating legacy systems to SOA [4].

This paper presents the findings of a case study of the migration of a large scale legacy system from a Dutch bank to a service-oriented architecture. For reasons of confidentiality, hereinafter the bank is referred to as "NedBank". The paper describes a 4-phase migration process that is used in NedBank. For each phase, the paper identifies the benefits of using particular techniques/methods within that phase, best practices that helped to achieve success, and possible challenges that were faced during migration. Based on these observations, the paper presents the lessons learned from the case study. The findings of the paper not only emphasize the benefits of using reverse engineering techniques to facilitate the migration process, but also urges academia to pay attention to business and organizational aspects. The business and organizational aspects include governance of the migration process, early involvement of the existing technical staff, and knowledge harvesting of the system.

The paper is structured as follows: Section II discusses related work; Section III explains the research approach and current technology landscape of the payments domain of NedBank; Section IV presents the migration process and discusses benefits, best practices and challenges faced during the migration; Section V analyzes and presents the findings as lessons learned. In Section VI, the paper concludes with some potential future work.

II. Related work

De Lucia et al. [5] describe an approach and tools to migrate legacy applications to web applications. Sneed [6], [7] has contributed several wrapping techniques to migrate COBOL applications to SOA. A plethora of research has been reported on migrating legacy applications to SOA. They are reflected in the survey [4]. However, considerably fewer real world case studies of legacy to SOA migration are reported. Nasr et al. [8] describe two large scale industrial case studies of legacy to SOA migration; Colosimo et al. [9] present an empirical study of legacy migration in Italian companies; Kokko at al. [10] report on SOA adoption process in nine Finnish organizations.

The use of reverse engineering techniques in software evolution has been extensively researched. Various research roadmaps and surveys (e.g., Bennett et al. [11], Muller et al. [12], Canfora et al. [13]) have been presented. As per the interest of this research, extracting program quality metrics has been reported in [14], [15] and details of use of software visualization in reverse engineering & re-engineering has been reported in a survey by Koschke [16]. Data-intensive legacy system migration has been reported by Henrard et al. [17].

III. Research Background

The current research has adopted an exploratory case study method [18], primarily reporting on how the migration is carried out and seeking new insights about which activities are performed during migration. Data collection in this case study is performed based on the participant observation method [18], wherein two of the researchers were directly involved in the migration project. The data collection included the following: (i) *consulting documentation* to identify the need for and goals of the migration process; (ii) *workshops* & *informal discussions* to discuss the progress of the migration in real time, and (iii) *semi-structured interviews* to understand various aspects of the migration process. In total six semi-structured interviews were conducted. The interviews were conducted in English and lasted between 60-120 minutes. Prior to the interviews, each expert was introduced to an interview protocol, a document detailing the objectives of the interview with some sample questions, and a glossary of the technical terms to attain a common understanding.

Research Context: NedBank is one of the largest banks in the Netherlands with more than 900 branches worldwide. Triggered by the Single Euro Payment Area (SEPA) initiative of the European Union, NedBank started the migration of its core banking systems under a project that started in 2010 and is expected to end in 2018. The main objective of this project is to renovate and migrate its legacy systems to SOA. The estimated cost of the project is 600M Euro. The project is subdivided into 6 different portfolios: channel support, payments, current accounts, customer reporting, counter, and sales & product agreements. After an initial investigation, this research is scoped to the migration of the payments domain because of the following two reasons: (i) the subsystems within the payments domain are diverse with respective to programming languages, hardware and operating systems in use, and (ii) the payments domain is of prime importance in the day-to-day operation of the banking business.

The payments domain is responsible for the overall management of banking transactions including foreign transactions, and interest & cost calculation per transaction of NedBank customers. The subsystems within the payments domain are considered to have high impact on the business of NedBank and have a high priority within the banking system. The payments domain was one of the first domains to adopt automatization in NedBank. Over the years, the systems of the payments domain have been subjected to frequent changes which have resulted in a "spaghetti architecture" [19] posing long-term problems such as increased complexity, inflexibile to changes and evolution, and increasing maintenance and running costs. Currently, the payments domain consists of five major legacy subsystems as detailed in Table I.

TABLE I
Details of the subsystems in the payments domain

Subsystem	Language	Platform	LOC
CalculateInterest	COBOL	IBM Z/OS	401,761
ForeignAccount	COBOL	HP Tandem	2,193,570
BalanceCheck	COBOL	HP Tandem	817,882
AccountAgreement	COBOL	IBM Z/OS	529,055
ReportCustomer	COBOL	HP Tandem	587,519

To better understand the basic working principles of and dependencies between these systems, a use case is described in which a customer is created and (s)he withdraws money from an Automated Teller Machine (ATM). Figure 1 depicts a high level sequence diagram, in which every directed edge between two subsystems implies a coupling between the two.

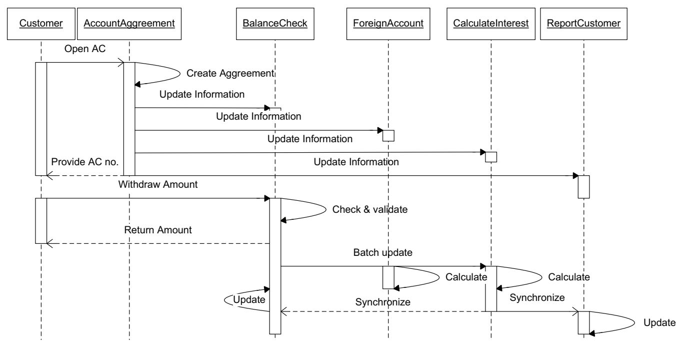


Fig. 1. Sequence diagram depicting coupling within the payments domain

A new contract for opening an account is created in a Siebel-based sales environment in one of the local branches. During the account opening process, Siebel requests several *AccountAgreement* services in order to get an account number and to create agreements for the customer. The account and agreement creation are processed in real-time. Upon opening an account for a customer, the other four subsystems (*BalanceCheck*, *ReportCustomer*, *ForeignAccount* and *CalculateInterest*) are updated accordingly. When the customer withdraws money from an ATM, initially the request is validated with the agreements stored in the *BalanceCheck* subsystem and the withdrawn amount is reserved from the customer's account. Such individual banking transactions are stored in a flat file and at the end of the day, the flat file is updated with the transactional information from the *ForeignAccount* subsystem: a subsystem responsible for recording the foreign transactions. Afterwards, the flat file is processed by the *CalculateInterest* subsystem that is responsible for interest, commission and cost calculations, and synchronizing the updated records to the other subsystems.

From the information in Table I, it is obvious that the subsystems of the payments domain are combined with heterogeneous IT infrastructures with variations in the COBOL dialects used, and the hardware platforms on which they operate. The subsystems range from internally developed subsystems like *CalculateInterest* to third party built-in packaged subsystems such as *ForeignAccount*. The subsystems are efficient in terms of performance, capable of effectively analyzing, processing and synchronizing millions of records. Nevertheless, to achieve such performance, various features within the subsystems are duplicated and/or updated in ad-hoc manner, increasing system complexity. The increase in complexity has now become a bottleneck to the changeability of the subsystems within the payments domain.

Additionally, based on our observations (interview, documentation, workshops and informal discussion), the main business goals of the NedBank upon migrating to a SOA are (i) accelerating time-to-market, (ii) reducing costs in the payments domain, (iii) transparency in ownership & governance of the products, and (iv) preventing knowledge erosion.

IV. The Migration Process

In this section, we explain the legacy to SOA migration process of the payments domain. Due to the complexity and tight coupling between the subsystems of the payments domain, the activities within the migration process are performed in a phased, controlled manner based on the business priorities. The migration approach consists of the following four phases, being: (i) Forming a migration program management committee, (ii) Developing a logical target-architecture, (iii) Analyzing the gap, and (iv) Realizing the migration.

A. Forming a migration program management committee

The legacy to SOA migration process needs to be capable of addressing various kinds of issues including business, organizational and technical issues [8]. The migration process involves a long term investment of resources and is aimed at conducting a large-scale migration by performing activities with minimal dependencies and maximal parallelization. Thus, to establish a suitable planning and management, a governing body, the *Program Management Committee*, was created. It includes various stakeholders representing senior management officials, business architects representing the different business units, software architects and technical managers, external consultants and application developers. The committee is divided into the following teams with specific responsibilities: a *Steering Committee* to develop a strategic policy for the migration; a *Core Team* to develop the business-IT alignment strategy; a *Program Management* team to manage the payment portfolio; a *Business Change Management* team to ensure proper alignment of business goals with the IT architecture; and an *Architecture Board* to develop an architectural governance within the payments domain. The role of the *Business Change Management* and *Architecture Board* is crucial, in particular, in developing and executing the migration process, aligning the business goals with the architectural requirements, and coordinating architectural priorities inline with the business goals.

Benefits, challenges and best practices: A legacy to SOA migration is a multifaceted process that involves technical, organizational and business issues [20]. To manage such a multifaceted process, a central governing body with suitable governance of the entire migration process is indispensable. Needless to say, a legacy to SOA migration is a complex and challenging process and any failure can threaten the success and fortune of an enterprise [21]. In particular, software failures in the financial domain not only cost millions but also decrease customer confidence[1]. In this migration process, the formation of the *Program Management Committee* has suitably fulfilled the need of such a governing body and hence, contributes towards a successful migration. The teams within the *Program Management Committee* have clear responsibilities such that any unpredicted changes were systematically resolved. For instance, any Request For Change (RFC) is primarily resolved by the *Business Change Management* and *Architecture Board* unless the RFC has high business priority and higher estimated cost than a chosen threshold value. Then, the RFC is forwarded with recommendations from the *Business Change Management* and *Architecture Board* to the *Core Team* and to the *Steering Committee* for further considerations.

Realizing that a large scale migration to SOA is not only a technical endeavor, the existing knowledge within the technical staff need to be utilized. The involvement of technical staff (legacy system developers and maintainers) in the committees facilitated the knowledge transfer to the migration team. It is a recurring phenomenon that the technical staff is hesitant to share knowledge due to the fear that their expertise may become redundant due to migration. This phenomenon was countered here by involving the technical staff to actively participate in the migration process.

B. Developing a logical target-architecture

Initially, a logical target-architecture conforming to the business goals was developed. A logical target-architecture forms the organizing logic for business processes and IT infrastructure, in which the business components are contained. Developing a logical target-architecture that conforms to the business goal was not an easy task. To start with, a group of members from the migration project initially participated in a workshop to define a functional architecture: an architectural model that identifies features that contribute to achieving the business goals. The team members included business process analysts and business architects from the *Business Change Management* team along with software architects and application developers from the *Architectural Board*. Various other external consultants and experts from financial software vendors also participated in the workshop. Together they provided the initial blueprint of the functional architecture. Following

[1]IT failure of Royal Bank of Scotland (RBS): http://goo.gl/xpDjy

the first workshop, three more workshops were conducted that resulted in identifying various business components to realize the initial functional architecture as shown in Figure 2. The identification of the business components, referred to as "componentization", was one of the initial activities to realize potential candidate services. The notion of componentization is a way to construct a business component, which corresponds to a feature contributing towards a business goal.

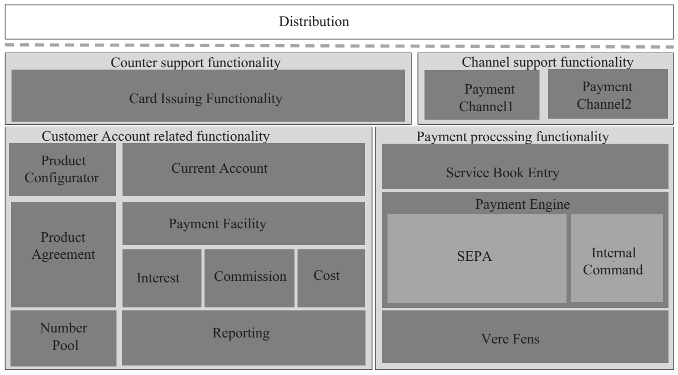


Fig. 2. Logical Target Architecture

Previously, such features were scattered over various subsystems. For instance, the "calculate interest" feature was previously found in two subsystems: (i) *CalculateInterest* and (ii) *BalanceCheck*. Earlier, "product agreements" feature was also distributed over various subsystems. In the logical target-architecture, related features are gathered within one logical unit to ensure that architecture governance is easy, and to minimize the product gaps within the payments domain.

Benefits, Challenges and Best Practices: Identifying business components representing potential candidate services in legacy to SOA migration is a challenging task. In this migration process, a goal-service modeling approach, proposed by Arsanjani et al. [22], is followed. A goal-service modeling approach is used to componentize the business component because it ties services to the business goals. Each identified candidate service was prioritized based on its business value. Additionally, a catalogue for each business component was created, indicating the degree of reusability by other components and possible functional dependencies (coupling) with other components in the logical architecture. Such a catalogue provides an overview of components whose migration can be performed independently, preferably in parallel with other relatively independent components and hence, maximizing parallelization of the migration process. In total, 44 different high level features were identified.

C. Analyzing the gap

The third phase of the migration process is to gather and determine the information about the legacy system features that can contribute to the realization of the logical target-architecture. The payments domain consists of a mix of many systems ranging from in-built COBOL system such as *CalculateInterest* to a third party packaged application such as *ForeignAccount*. There are not only variations in the COBOL dialect, but also in the running platform such as IBM Z/OS, HP Tandem Nonstop. Also, the documentation of most subsystems was outdated. An investigation of the documentation quality of the *CalculateInterest* system identified missing technical documentation (TD), limited finalized/approved documentation, and fairly good functional documentation (FD). However, the details of the TD and FD for features are still not complete. Furthermore, the technical quality characteristics such as coupling, maintainability, and duplication within the subsystems were still unknown.

As a starting point, all the subsystems of the payments domain were analyzed using source code analyzers to determine the program quality in terms of quality metrics including maintainability, module coupling, duplication and changeability. These quality metrics were derived using reverse engineering tools. Such quality metrics provided a better understanding of the technical qualities of the subsystems within the payments domain. Table II depicts an excerpt of the assessment results of the subsystems in the payments domain in which Maint. represents maintainability; Coup. represents coupling; Dup. represents duplication; Change. represents changeability and Test. represents testability metrics. Refer to [14], [15] for the details and explanations of these metrics, as they are out of the scope of this paper.

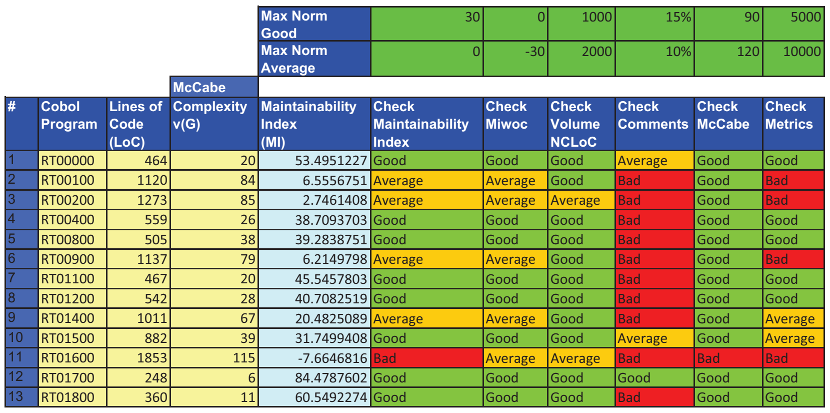
TABLE II
EXCERPT OF LEGACY ASSESSMENT RESULT

Name	#prog	Maint.	Coup.	Dup.	Change.	Test.
CalculateInterest	913	2.10	2.14	1.32	2.06	2.13
ForeignAccount	9249	1.07	2.47	1.24	1.95	1.80
BalanceCheck	2902	2.03	2.70	1.32	2.05	1.72
AccountAgreement	1364	2.45	3.69	1.34	2.33	1.77
ReportCustomer	918	2.25	3.04	1.23	2.24	1.82

Furthermore, to have an in-depth understanding of the technical qualities of each COBOL program, a detailed analysis was carried out for each subsystem using proprietary automated source code analyzers. Such a detailed analysis provided insights into individual COBOL programs within each subsystem. For instance, individual programs were categorized into good, bad and average based on their complexity. Figure 3 depicts an excerpt of a detailed program analysis derived from source code analyzers of the *CalculateInterest* COBOL programs. Due to space limitations, the detailed analysis is not presented in this paper, but anonymized reports of the *CalculateInterest* and the *ForeignAccount* are available[2].

As a part of the legacy assessment, a call dependency diagram of the subsystems was generated and analyzed based upon number of incoming call (NIC) and number of outgoing calls (NOC). As a result, numerous computationally intensive COBOL programs (with high NIC and high NOC) and core libraries (with high NIC) within each subsystems were identified, following the work of Geet & Demeyer [23].

[2]http://goo.gl/bwqnq



				Max Norm Good	30	0	1000	15%	90	5000
				Max Norm Average	0	-30	2000	10%	120	10000
			McCabe							
#	Cobol Program	Lines of Code (LoC)	Complexity v(G)	Maintainability Index (MI)	Check Maintainability Index	Check Miwoc	Check Volume NCLoC	Check Comments	Check McCabe	Check Metrics
1	RT00000	464	20	53.4951227	Good	Good	Good	Average	Good	Good
2	RT00100	1120	84	6.5556751	Average	Average	Good	Bad	Good	Bad
3	RT00200	1273	85	2.7461408	Average	Average	Average	Bad	Good	Bad
4	RT00400	559	26	38.7093703	Good	Good	Good	Bad	Good	Good
5	RT00800	505	38	39.2838751	Good	Good	Good	Bad	Good	Good
6	RT00900	1137	79	6.2149798	Average	Average	Good	Bad	Good	Bad
7	RT01100	467	20	45.5457803	Good	Good	Good	Bad	Good	Good
8	RT01200	542	28	40.7082519	Good	Good	Good	Bad	Good	Good
9	RT01400	1011	67	20.4825089	Average	Average	Good	Bad	Good	Average
10	RT01500	882	39	31.7499408	Good	Good	Good	Average	Good	Average
11	RT01600	1853	115	-7.6646816	Bad	Average	Average	Bad	Bad	Bad
12	RT01700	248	6	84.4787602	Good	Good	Good	Good	Good	Good
13	RT01800	360	11	60.5492274	Good	Good	Good	Bad	Good	Good

Fig. 3. Excerpt of a detailed program analysis of the *CalculateInterest* COBOL programs

Such programs were later manually investigated to locate features within the subsystems. Figure 4 depicts an excerpt of the generated call dependency diagram of the *CalculateInterest* subsystem in which the red-circled programs (RT23N, RT23M, RT20K and RT009), for instance, could potentially be core libraries of interest.

After the legacy assessment, an inventory of high level features available within the subsystems of the payments domain was created. The inventory of the features was created by consulting the available documentation and interviewing the technical staff of the subsystems. The latter method proved to be very useful and confirms that the knowledge residing within the organization is of the utmost importance. The inventory of the high level features was then analyzed by the *Business Change Management* and *Architecture Board* to determine the priority and business value. The features were then mapped to the logical components within the logical target-architecture via the gap analysis method [24]. The mapping was performed by focused workshops conducted for each subsystem in which business analysts, application analysts, consultants, developers and lead architects discuss and finalize the mapping of each subsystem. This mapping approach was effective such that the migration team not only identified the mappings, but also the dependencies within the high level features of the subsystems. Table III depicts an excerpt of the feature mapping of the *CalculateInterest* and the *AccountAgreement* to the logical target-architecture.

Benefits, Challenges and Best Practices: The "analyzing the gap" phase enabled the migration team to catalogue the existing features with the aim to maximize reuse features. In particular, the use of reverse engineering tools/techniques has facilitated understanding the current legacy assets, their technical qualities, and identifying the potential features based on call dependency diagrams. The "analyzing the gap" phase has not only been effective in identifying, prioritizing and determining the granularity of existing features, but also in determining which feature is to be reused. The legacy assessment activity contributed to identifying the technical program quality in terms of software metrics such as maintainability, module coupling, duplication, changeability. Such software metrics have been extensively used in the software evolution domain, for instance, to determine the reusability factor [6]. To better understand the individual programs within each subsystem, the program level quality metrics along with the program visualization in the form of a call dependency diagrams were generated using reverse engineering tools. Furthermore, the interview sessions with the technical staff of the payments domain proved to be extremely important. Needless to say, intimate knowledge of the existing resources is essential to a successful migration, and necessary steps should be taken to harvest and preserve such existing knowledge.

D. Realizing the migration

The payment domain of the bank has a heterogeneous IT infrastructure with some of the features being efficient and robust with respect to performance while others being rigid commercial off-the-self (COTS) applications. In such a scenario, relying on a single approach to realize migration is not a viable solution. Thus, the migration process made the following four explicit choices for realization:

1) Reuse and/or Upgrade: One of the key performance indicators of a bank is accuracy and efficient processing of voluminous financial transactions. In the payment domain, some of the features are highly robust in terms of accurate and efficient processing of transactions. Such features are either reused or upgraded based on their business value and the program quality characteristics derived in the "legacy assessment" of the "analyzing the gap" phase. For example, the *"calculate interest"* feature is reused. With regards to the *CalculateInterest* subsystem, one of the business analysts says "*The clear separation in the features of the CalculateInterest subsystem has eased our maintenance. Also, if we consider rebuilding or splitting the features then the impact will be very high– technically and economically and we are not sure if we can achieve the current performance. Thus, for the time being we decided to reuse the features of the CalculateInterest*".

2) Package Replacement: Numerous logical components within the logical target-architecture cannot be directly

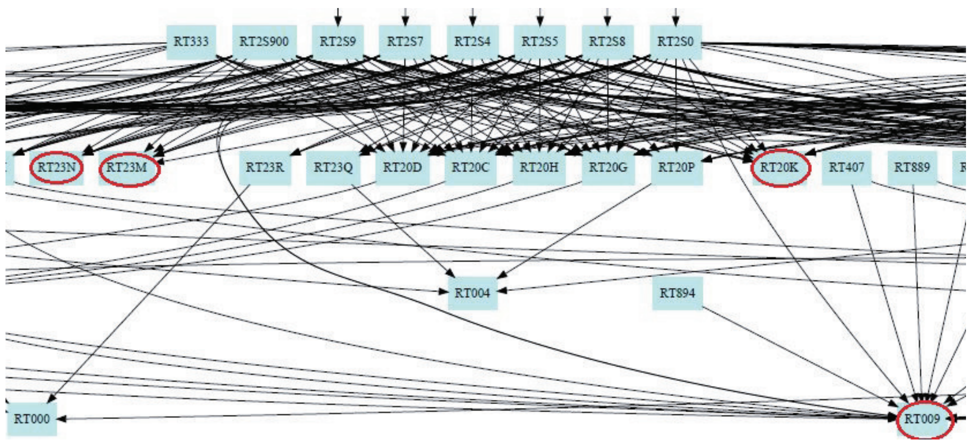


Fig. 4. Excerpt of a call dependency diagram of the *CalculateInterest* COBOL programs

TABLE III
EXCERPT OF FEATURE MAPPING TO THE LOGICAL TARGET-ARCHITECTURE

High Level feature	**Target Arch. Component**	**Priority**	**Remarks**
	CalculateInterest		
Register data	Bank Administration	High	–
Calculate interest	Interest	High	Merge international interest
Bank guarantee commission	Fees	Medium	–
Checkout coupon	Interest	Medium	To be included in the Interest logical component
	AccountAgreement		
Opening accounts/contracts	Product Agreement	High	Merge current agreements in current account
Account management	Number Pool	High	–
Managing data rate	Product Configurator	Low	Include tariff data from other components

mapped to existing features of the legacy applications. Thus, some of the components in the target-architecture are being replaced by a packaged solution. The decision to replace is reached by assessing the technical program qualities and economical feasibility of the feature. One of the examples of such a replacement is within the features of the *ForeignAccount* subsystem, which in itself is a third party packaged subsystem responsible for international payments. Thus, the features of the *ForeignAccount* subsystem are replaced by a packaged solution. An application architect says "*ForeignAccount is a package software with very limited documentation and reusing its features will lead to long-term maintenance problems in the future. So we decided to replace it with a packaged solution*".

3) Customized Replacement: With the introduction of the Euro currency, various laws and regulations within the payment domain have changed in the European Union. The bank has to comply with such changes. One of the business consultants emphasizes the importance of the SEPA stating that "*SEPA is one of the triggers for the renewal of the whole payments infrastructure. It is also one of the main business drivers for lowering cost. Such crucial features have to be custom–built so that its maintenance and upgrade in the future will be easy for us*".

4) Outsourcing: The final option is to outsource an entire feature to an external party for development. This option is chosen only if outsourcing contributes to the strategic objectives of Payments (such as cost reduction) and must fulfill the requirements as formulated in the outsourcing strategy-guidelines to ensure that outsourcing is done via strategic partners and only when no other option is viable. A lead architect explains the need of outsourcing as "*Features that are of low business value and can be developed cheaply are outsourced such as card authorization and card payments. This helps us to focus on the high priority features.*"

The selection of an appropriate realization option is based on various factors such as business value of the logical component, technical quality of the legacy assets, cost of implementation and the importance of ease of upgrading/updating.

Benefits, Challenges and Best Practices: Realization of the migration is the starting point of implementing the logical components. Realization is not only about deciding which programming language is to be used, but also the associated environment such as hardware and operating system. The other factor that contributes to the success of realization is the determination of the suitable granularity of the potential candidate services. In this migration project, there were predefined guidelines provided by the program management committee on deciding which realization option to use. The guidelines were developed by considering "external vs internal development" and "adoption of existing vs new technology". Figure 5 depicts the realization options based on development and technology adoption criteria.

A crucial criterion to determine the realization option was

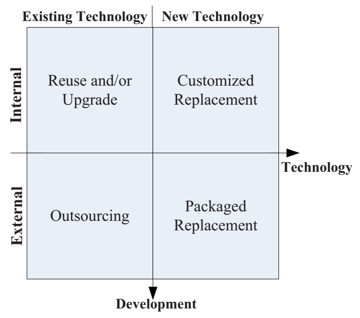


Fig. 5. Realization Choices

the business value and priority of the logical component. For instance, central to NedBank's business are the calculation of interest, commission and cost calculation features. The logical components encompassing these features are reused and/or upgraded from the existing ones. Upon deciding to reuse and/or upgrade, the technical qualities of the features are examined to estimate the migration time. It was not always simple to reuse or upgrade, particularly for third party packaged solutions that are not updated or supported by the vendor anymore. For instance, the vendor who developed COBOL running in HP Tandem Nonstop went through various mergers and acquisitions such that the infrastructure is hardly updated and maintained. In such cases, a suitable packaged replacement is preferred. Any new logical component with high business value falls under the "Customized Replacement" option. A low priority logical component is outsourced to a third party upon strictly fulfilling the "outsourcing" criteria.

V. Lesson Learned

A. Implications of reverse engineering techniques

The reverse engineering techniques that are used in this migration project had a significant role in finding the facts of the current legacy programs and subsystems. In particular, the use of such techniques in the "analyzing the gap" phase to obtain various metrics and call dependency graphs not only facilitated the creation of an inventory of the current assets, but also identified computationally intensive COBOL programs. In addition to creating an inventory of the current assets, using reverse engineering techniques has the following two implications:

(i) *Assisting in selecting the realization approach*: The use of reverse engineering techniques has strongly facilitated the selection of the realization approaches in the migration process. The migration team used the program quality metrics generated by the reverse engineering tools to estimate the complexity of the programs. For example, in Figure 3, the COBOL program RT01600 has high McCabe complexity and a negative maintainability index (MI). Hence, *RT01600* is a potential candidate of replacement unless it has a low business priority. Similarly, programs with *good* and *average* index have potential for reuse. In case of identifying candidate COBOL programs for services, the generation of call dependency graphs of the subsystems was helpful. For instance, based on the work of Van Geet et al. [23], the red-circled programs (RT23N, RT23M, RT20K and RT009) of Figure 4, were COBOL programs of interest as they have high number of incoming calls (NIC). Thus, upon generating and analyzing the call dependency graph those programs were investigated by the existing programmers to identify their functionalities.

(ii) *Knowledge harvesting*: Most of the documentation of the subsystems was either outdated or incomplete. With the results of the reverse engineering techniques, a considerable amount of new information about the programs was identified that were even unknown to the current maintainers of the subsystems. For instance, 599 out of 21085 copybooks are not used by the *"CalculateInterest"* subsystem. This finding was a surprise to the current maintenance team of the subsystem. Additionally, the flow graphs were generated to understands the overall flow of the programs within the subsystem. These artifacts helped to update the documentation.

B. Adopting a pragmatic realization approach

In the realization phase of the migration process, a pragmatic approach for executing the migration is adopted in which the choices are based on various factors such as business value, business priority, and the technical qualities of the features. The initial choice of reusing the existing functionalities is one of the notable benefits claimed by the proponents of SOA for leveraging existing assets. However, in a large scale legacy application, reuse is not always feasible. Therefore, the migration process of NedBank suitably adapted other possible realization methods for a successful migration. Additionally, for large scale legacy applications that include heterogeneous IT infrastructures (diverse programming languages, various hardware and operating systems) there is no silver bullet solution to realize the migration process. For a successful migration process, any approach can contribute to the success of the migration, provided that the approach is well defined, and suited for the enterprise.

C. Emphasizing organizational and business perspectives

Migration from legacy to a SOA environment is not only a technical endeavor, but also involves significant issues from the organizational and business perspective. Particularly in the case of legacy systems having no or outdated documentation, early involvement of existing technical staff in the migration process is proven to be useful. The involvement of the technical staff facilitates the knowledge transfer to the migration team. The synergy between the technical staff and the migration team that was observed in this migration process was one of the key factors contributing towards the success of the migration. Equally important is the focus of the *"Program Management Committee"* in the business-IT alignment that facilitated the migration to achieve the business goals. An important lesson learned is that technical staff tends to resist change because they fear that their expertise and professional experience with legacy systems may become redundant due to the introduction of SOA. Therefore, it is important to involve the technical staff from the start of the migration process and provide necessary training to adapt to new technology. In the current migration process, the formation of various teams under the "*Program Management Committee*" has actively involved the technical staff whose knowledge about the legacy systems have proven to be of significant importance.

D. Harvesting knowledge to prevent knowledge erosion

Apart from available documentation, existing knowledge in the form of skills and experience within the technical staff is one of the most important assets. Over the years, such knowledge and skills become scarce resulting in knowledge erosion due to factors such as ageing, and staff changing jobs. Hence, suitable strategies such as conducting and archiving interviews, and initiating knowledge transfer programs via training should be undertaken to harvest and preserve such existing knowledge. In the migration process of NedBank, the involvement of the technical staff in the migration process has significantly helped in knowledge harvesting and preservation while creating inventories of the high level functionalities of the subsystems. Some of technical staff were interviewed and focused training programs were organized to facilitate knowledge transfer. Additionally, the existing documentation was updated or created in case if no documentation exists.

VI. Conclusion

In this paper, we present the findings of a case study of a large scale legacy to SOA migration process within the payments domain of a Dutch financial institution. The paper presents the business drivers that initiated the migration, and describes a 4-phase migration process. The migration process equally focuses on the technical and the business & organizational aspects of the migration. The migration process starts by forming a "*Program Management Committee"* for proper governance. Then, a logical-target architecture is developed based on the concept of componentization. The business components within the logical target-architecture are aligned to support the business goals. The logical target-architecture is then mapped to the existing legacy features using a gap analysis method. Such a gap analysis suitably supports the potential of reusing the legacy features without significant changes to the legacy systems itself– one of the key promises of the SOA. The migration is then realized following the pragmatic realization options of the migration process.

The paper presents an industrial report detailing how reverse engineering techniques were employed to facilitate a large scale legacy to SOA migration process. It illustrates the pain of the practicalities and under-emphasized aspects of a large scale migration project, and should be considered a call-to-action for computer scientists to study the project environment, both from a business and organizational point of view, as much as they study the technical aspects of migration.

As to future research, we aim to exploit model-driven modernization approaches to extract features from the subsystems. Another interesting research area is to investigate how to automatically translate legacy applications to a modern language.

References

[1] M. van Sinderen, "Challenges and solutions in enterprise computing," *Ent. Inf. Sys.*, vol. 2, no. 4, pp. 341–346, 2008.

[2] K. Bennett, "Legacy systems: coping with stress," *IEEE Soft.*, vol. 12, no. 1, pp. 19–23, 1995.

[3] M. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-oriented computing: State of the art and research challenges," *Computer*, vol. 40, no. 11, pp. 38–45, 2007.

[4] R. Khadka, A. Saeidi, A. Idu, J. Hage, and S. Jansen, "Legacy to SOA evolution- a systematic literature review," in *Migrating Legacy Applications: Challenges in Service Oriented Architecture and Cloud Computing Environments*, A. D. Ionita, M. Litoiu, and G. Lewis, Eds. IGI Global, 2012, pp. 40–71.

[5] A. De Lucia, R. Francese, G. Scanniello, and G. Tortora, "Developing legacy system migration methods and tools for technology transfer," *Soft.:Pract. Exp.*, vol. 38, no. 13, pp. 1333–1364, 2008.

[6] H. M. Sneed, "A pilot project for migrating COBOL code to web services," *STTT*, vol. 11, no. 6, pp. 441–451, 2009.

[7] H. M. Sneed, "Migrating from COBOL to Java," in *International Conference on Software Maintenance*. IEEE, 2010, pp. 1–7.

[8] K. Nasr, H. Gross, and A. van Deursen, "Realizing service migration in industry–lessons learned," *J. Soft. Maint. Evol.*, 2011.

[9] M. Colosimo, A. D. Lucia, G. Scanniello, and G. Tortora, "Evaluating legacy system migration technologies through empirical studies," *Inf. Soft. Tech.*, vol. 51, no. 2, pp. 433–447, 2009.

[10] T. Kokko, J. Antikainen, and T. Systa, "Adopting soa–experiences from nine finnish organizations," in *13th European Conference on Software Maintenance and Reengineering*. IEEE, 2009, pp. 129–138.

[11] K. H. Bennett and V. T. Rajlich, "Software maintenance and evolution: a roadmap," in *Future of Software Engineering*. ACM, 2000, pp. 73–87.

[12] H. A. Müller, J. H. Jahnke, D. B. Smith, M.-A. Storey, S. R. Tilley, and K. Wong, "Reverse engineering: a roadmap," in *Future of Software Engineering*. ACM, 2000, pp. 47–60.

[13] G. Canfora and M. Di Penta, "New frontiers of reverse engineering," in *Future of Software Engineering*. IEEE, 2007, pp. 326–341.

[14] I. Heitlager, T. Kuipers, and J. Visser, "A practical model for measuring maintainability," in *6th International Conference on Quality of Information and Communications Technology*. IEEE, 2007, pp. 30–39.

[15] Y. Kanellopoulos, C. Tjortjis, I. Heitlager, and J. Visser, "Interpretation of source code clusters in terms of the iso/iec-9126 maintainability characteristics," in *European Conference on Software Maintenance and Reengineering*. IEEE, 2008, pp. 63–72.

[16] R. Koschke, "Software visualization in software maintenance, reverse engineering, and re-engineering: a research survey," *J. Soft. Maint. Evol.*, vol. 15, no. 2, pp. 87–109, 2003.

[17] J. Henrard, D. Roland, A. Cleve, and J.-L. Hainaut, "An industrial experience report on legacy data-intensive system migration," in *International Conference on Software Maintenance*. IEEE, 2007, pp. 473–476.

[18] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Emp. Soft. Eng.*, vol. 14, no. 2, pp. 131–164, 2009.

[19] R. Heckel, R. Correia, C. Matos, M. El-Ramly, G. Koutsoukos, and L. Andrade, "Architectural transformations: From legacy to three-tier and services," in *Software Evolution*, T. Mens and S. Demeyer, Eds. Springer, 2008, pp. 139–170.

[20] S. Murer, B. Bonati, and F. Furrer, "Very large information system challenge," in *Managed Evolution*. Springer, 2011, pp. 3–34.

[21] R. N. Charette, "Why software fails," *Spectrum*, vol. 42, no. 9, pp. 42–49, 2005.

[22] A. Arsanjani, S. Ghosh, A. Allam, T. Abdollah, S. Ganapathy, and K. Holley, "SOMA:A method for developing service-oriented solutions," *IBM Sys. J.*, 2008.

[23] J. Van Geet and S. Demeyer, "Lightweight visualisations of COBOL code for supporting migration to SOA," *Symposium on Software Evolution*, vol. 8, 2008.

[24] G. Lewis, E. Morris, and D. Smith, "Service-oriented migration and reuse technique (SMART)," in *Workshop on Software Technology and Engineering Practice*. IEEE, 2005, pp. 222–229.