2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering

A Systematic Framework for Modernizing Legacy Application Systems

Timothy C. Fanelli[1,2], Scott C. Simons[3], and Sean Banerjee[1]

[1]Department of Electrical and Computer Engineering, Clarkson University, Potsdam, NY, USA
{tfanelli, sbanerje} @clarkson.edu
[2]ZBL Services, Inc. - tfanelli@zblservices.com
[3]IBM, z Systems Workload Modernization, Poughkeepsie, NY, USA - scsimons@us.ibm.com

***Abstract*—Researchers estimate that $180 - 200$ billion lines of legacy code are still in use today. Legacy application system modernization is driven by an organization's desire to remain agile to changes, reduce operating costs and gain a clearer understanding of their application system infrastructure. In this paper we present a systematic modernization framework for legacy systems. Our framework considers the application code, information system and infrastructure as distinct entities which can be modernized independently. As a result, our approach can be applied sequentially to a single program module, or in parallel across multiple program modules. We validate our framework through an industrial case study involving a large, multinational financial services institution. Our case study results show that through modernization the organization reduced its cost to the sponsoring users by 80%, lines of code by 89%, operational complexity by 99%, and achieved an estimated 66% reduction in the man-hours required for time to market using the new application functions.**

I. Introduction

A legacy application system is defined as a system comprised of outdated software and / or hardware. However, software and hardware aging are not the only drivers for systems turning legacy. The loss of the system development team, increase in code complexity, and inherited and commissioned systems are some of the other reasons why systems turn legacy [1, 2]. Despite the challenges of operating legacy systems, they are also often perceived as business critical, proven and reliable due to their long operational history [3, 4]. Researchers estimate that $180 - 200$ billion lines of legacy code are still in use today [3, 4]. A recent survey conducted on 26 industrial practitioners revealed that almost half of their legacy application systems were written in COBOL [3, 4].

The desire to modernize legacy systems is driven by organizations wanting to remain agile to changes, reduce operating cost and obtain a clearer understanding of their system infrastructure [2, 5]. Modernization is also often necessitated by a skills shortage in legacy systems [6]. Successful modernization enables an organization to embrace modern development paradigms, reduce the time to market, obtain code clarity by removing redundancies, increase code confidence and quality [3, 4]. However, the time and cost associated with modernization, fear of data loss, lack of knowledge of the legacy system, difficulty in extracting business logic, predicting return on investment of the modernization process and internal resistance to change can quickly stall attempts at modernization [3, 4, 7, 8, 9].

In this paper we present a systematic framework for modernizing legacy application systems. Our framework is novel as it considers the application code, the information system, and the infrastructure as distinct entities which may be modernized independently. Our framework has been validated through an industrial case study involving a large, multinational financial services institution. The institution has over $40,000$ employees and over USD $2 trillion in assets under management.

Our contributions are as follows:

1) A systematic framework for modernizing legacy application systems
2) An industrial case study using a financial services institution to validate the feasibility of the modernization approach
3) A baseline industrial case study using a wireless communication service provider to demonstrate how subject matter experts drive our modernization strategies

The rest of the paper is organized as follows: In Section II, we summarize the related work in modernization and the limitations of existing approaches. In Section III, we define our systematic modernization framework. In Section IV, we validate our approach through a case study on a large, multinational, financial services institution. In section V we explore the limitations of our approach. We conclude the paper in Section VI by providing directions for future research.

II. Related Work

Legacy modernization techniques are broadly categorized as either: big bang, database first, database last or hybrid [10]. The big bang approach performs a complete redevelopment of the legacy system using modern software, database and hardware systems. The database first approach migrates user data to modern database management systems, thereby allowing users immediate access to the data. However, this approach requires both the legacy and target system to remain operational and interoperable through gateways [11]. Data inconsistency issues makes this approach infeasible in mission critical systems.

These approaches are either undertaken as a black box or as a white box methodology [12]. Black box methods are often based on wrapping, to expose modern interfaces around legacy

978-1-5090-1855-0/16 $31.00 © 2016 IEEE
DOI 10.1109/SANER.2016.40

program modules [12, 13]. Black box wrapping approaches are often taken as they allow rapid integration and reuse of legacy program modules. By contrast, white box approaches require subject matter expertise of the legacy program modules and precede more in depth re-engineering of the solution [12].

The above approaches are only applicable to fully decomposable systems. In reality, legacy application systems are seldom fully decomposable. In [14] the authors propose a general set of migration rules. The approach analyzes the legacy system and identifies portions that contain essential data and business logic. The target, modernized, system is developed as a standalone system, however it is unclear how and when the transition from the legacy to the target system should occur. The DARWIN system [15] provides incremental approaches for migrating decomposable, semi-decomposable and non-decomposable systems. The incremental approach ensures any failures in the migration process can be recovered by reapplying the methodologies only on the failed step.

The RENAISSANCE approach [16, 17] allows the user to re-engineer a system as opposed to completely replacing it. A user choosing to re-engineer the system will select one of four options: revamp the existing interfaces, restructure the internal system without affecting the external interfaces, rearchitecture the system by migrating to a modern hardware framework, or redesign by incorporating some artifacts of the legacy system. The Butterfly approach [5], [18] also requires both the legacy and transformed system to remain operable during migration, however interoperability is not required, thus eliminating the complexity of gateways. The Butterfly approach focuses on data migration, which creates complexities in the migration process and introduces unnecessary delays.

Limitations of Existing Approaches:
Current modernization approaches consider the legacy system as an atomic candidate for modernization, containing an information system, applications layer, and a hardware platform. This all encompassing view of modernization involves not just the design and implementation of the application code, but also data migration and infrastructure changes as well [19]. These big bang approaches are too risky to apply to any business critical system. Incremental methodologies, such as DARWIN, attempt to mitigate these risks, but still incur significant architectural complexities and integration challenges [5, 10, 11, 15].

In database-first approaches, data architecture and migration is addressed before application and interface design. An organization's solution portfolio often consists of many legacy application systems which rely on the same data. This increases the cost, time, and risk involved in existing iterative frameworks as persistence gateways must be incorporated into each application sharing the data within the portfolio [5, 11].

Black box wrapping approaches neglect the underlying implementation of the program modules, and are inappropriate for software systems under ongoing maintenance, or for which an information system or platform migration is desired.

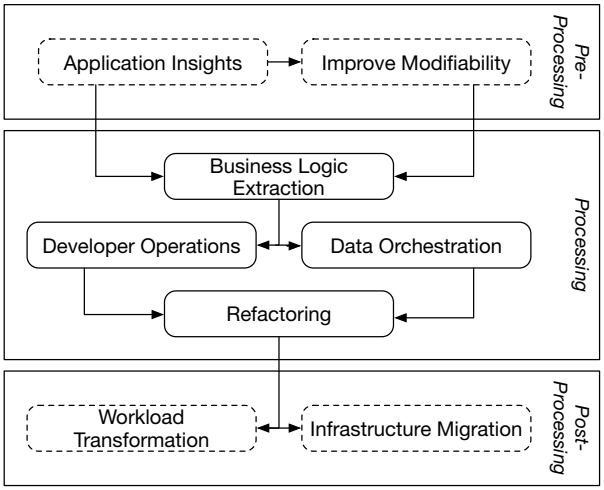

Fig. 1. A systematic framework for legacy program module modernization

III. Modernization Framework

A. Overview

Our modernization framework provides a phased approach to modernization which is either applied sequentially to any one program module, or in parallel across multiple program modules, comprising the overall software system. Our framework is novel as it considers the application code, the information system, and the infrastructure as distinct entities which are modernized independently; and places strong emphasis on the use of appropriate subject matter expertise. We focus specifically on the incremental transformation of the application system, and leave the hardware and information system changes as subsequent, simplified tasks. In its modernized state, the user has regained knowledge about the design and implementation of the application system, increased the agility of implementation, adopted known and well understood design patterns, and improved the code quality.

The modernization approach described is applicable to any structured, procedural language implementation, such as COBOL, PL/I and Assembler. The target language of modernization is any object-oriented language, such as Java, C#, or C++.

The framework consists of three phases: Pre-Processing, Processing and Post-Processing as shown in Figure 1.

B. Pre-Processing

The Pre-Processing phase consists of two optional steps: Application Insights and Improve Modifiability. These steps are designed to prepare the program modules for transformation. During Pre-Processing, the existing subject matter experts and sponsoring users of the legacy application are critical to the quality of the output, which drives the Processing phase.

1) Application Insights: Application insights is the application of code and system analytics to gain insight into the requirements, design, and implementation of the legacy software system. In our industrial case study, a legacy application was

identified in production that was last compiled in 1986. That application had not been in active maintenance or development for over 25 years. Much of the original knowledge about the application has been lost, both through attrition of the development staff responsible for it, and through data loss events associated with lack of digitized and formal requirements tracking systems. Application insights allows the organization to rebuild some of the lost domain knowledge about the system under investigation.

2) Improving Modifiability: The purpose of this step is to identify and separate business and program logic from data orchestration. Within the legacy application, non-destructive edits are made to identify source code artifacts which map to business requirements. This facilitates Business Logic Extraction during the Processing phase, and allows the legacy application subject matter experts to build in depth expertise about the business requirements.

3) Outcomes: Pre-Processing produces two outcomes. First, we have recreated, or augmented, supporting requirements and design artifacts of the existing legacy application. Second, the subject matter experts, in working closely with sponsoring users, have acquired additional expertise in the business requirements of the application.

Both these outcomes are critical to the success of the Processing phase. The requirements and design artifacts are used to drive Business Logic Extraction. Furthermore, the legacy application system subject matter experts become advisers to the design and implementation team responsible for the modernized application.

C. Processing

During the Processing phase, we address both the implementation of, and development culture governing, the program modules within the application system. Often, the subject matter experts responsible for the legacy application system are less experienced in the target language or platform of modernization. The success of the modernization effort, therefore, relies on the availability of target subject matter experts who will take over the design, implementation, and maintenance of the modernized application. In our industrial case study we demonstrate how subject matter experts drive modernization strategies.

1) Business Logic Extraction: The first step in the language transformation is the extraction of business logic from program modules. Business Logic Extraction is best performed at a requirements or functional specification level, rather than method by method within the program modules. However, this is often idealistic and difficult to accomplish, particularly when Pre-Processing is incomplete or omitted.

We focus this step strictly on the business logic contained within a program module, rather than program flow or data access code. In doing so, we enable increased reuse of the business logic implementation [20]. During development, we require the adoption of test-driven development methodology, beginning a cultural shift within the development organization necessary to the success of adopting developer operations in this phase.

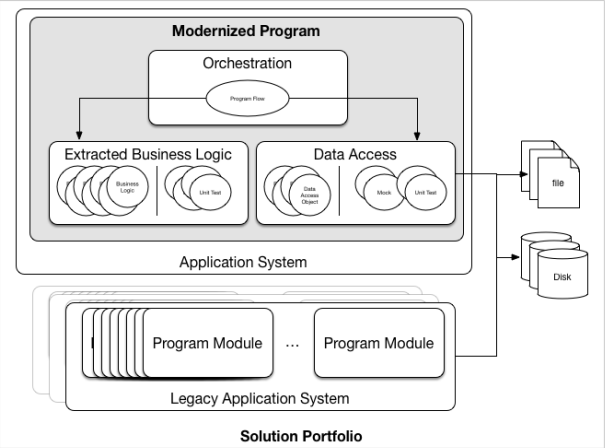

Fig. 2. Structure of a solution portfolio after modernization of one or more legacy application systems

Business logic in the legacy application is replaced by an invocation to the extracted business logic. This replacement simplifies what remains of the legacy application system, enables black box testing against the extracted business logic, and provides a continuous integration of the modernized asset.

2) Developer Operations: During business logic extraction, we require the adoption of a test-driven development methodology. In addition to increased code confidence [21], this also begins a cultural shift within the organization to adopt agile development and project management methodologies. Developer operations increases the productivity of development, decreases time to market for change, and results in more accurate project estimates and projections over time [3].

3) Data Orchestration: With the business logic successfully extracted from the legacy program module implementation, the primary responsibility of what remains in the original source is program and record orchestration: the input-output logic, record augmentation, and data validation routines that prepare the data for processing.

Test-driven development is also required during this step of the transformation, for reasons discussed earlier.

4) Refactoring: The final step in transformation involves refactoring the new program implementation to adopt known design patterns, best practices, and style guidelines resulting in increased readability and maintenance, and simplified design of the program.

Refactoring the design of an application is a normal part of maintenance to increase reusability of the modernized application. This is particularly true when the Processing phase when the legacy program's subject matter experts have a high degree of influence over the design of the target implementation. This often results in non-standard coding styles and the introduction of procedural workflow structures into object-oriented language environments.

5) Outcomes: As shown in Figure 2 after modernization of a legacy application, a solution portfolio now consists of both legacy application systems, and modernized application systems, which can rely on the same underlying data. The

modern application system is fully decomposable, simplifying its on-going maintainability and reuse.

The subject matter experts responsible for the legacy program module now hand off ownership of the modernized program module to the new team for ongoing maintenance. Having served as advisers during Processing, they gain insight and experience into the modernization process. These subject matter experts are now available to continue the modernization effort on other program modules or application systems within the solution portfolio.

D. Post-Processing

The Post-Processing phase comprises of any future end-state for broader transformation beyond the application itself, including information systems and hardware infrastructure. Infrastructure Migration and Workload Transformation are shown as two very common examples, in which workload is migrated to on- or off-premises cloud infrastructures, or consumed in online transaction processing workloads rather than batch.

IV. Case Study

Our framework is validated through industrial case study involving a multi-national financial services firm, with $40,000$ employees, and over $2 trillion USD in assets under management. In this section, we will discuss the organization's reasons for modernization, including their business value drivers, and provide an overview of their results.

A. Background

The organization's infrastructure cost is directly correlated to their utilization metrics. Peak utilization is measured over a 4 hour rolling average, and determines the licensed capacity of the infrastructure. This cost is recovered from sponsoring users at a fixed rate of C_1. C_1 represents the dollar amount per CPU service unit. A service unit is assumed to be a CPU second for simplicity.

The infrastructure vendor incentivizes adoption of Java by omitting the Java Virtual Machine from the measured utilization. This incentive allows the organization to bill sponsoring users a lower rate for Java applications: C_2 which is approximately $\frac{C_1}{8}$, providing strong financial drivers to fund modernization efforts.

The organization also faces a skillset shortage associated with COBOL development talent, legacy development paradigms, and an outdated development toolchain. Developer operations, agile development methodologies, modern development environments, and continuous delivery pipelines are further value drivers for modernization of legacy application systems. Several prior modernization efforts utilizing our baseline approach failed to demonstrate financial incentive for funding future initiatives.

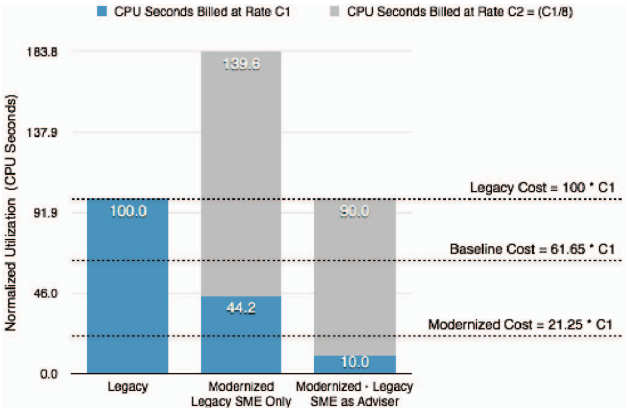

Fig. 3. Utilization measurements and cost of a modernized application system, comparing use of only legacy subject matter experts (wireless telecommunication provider) with use of legacy subject matter experts as advisers to target

B. Application Overview

The legacy application is a high volume, low latency, transactional batch workload that executes daily, processing business data generated during normal business. The organization uses this system internally, and in a software as a service model (SaaS) to other financial intuitions. A copy, modify, deploy pattern had been used to on board service customers, resulting in unmanageable maintenance overhead. The application system had grown to contain over $900,000$ lines of COBOL code, comprising 850 program modules orchestrated by approximately 5000 individual batch jobs. It takes the organization between 4 and 6 months to on-board a new service consumer. This model limited the organizations ability to scale the solution with the growth of the business, as maintenance and operational complexity increased with each new service consumer.

C. Baseline Predictions

To baseline our approach, we performed a prior case study with a wireless telecommunication provider supporting over 130 million subscribers. In this modernization approach we applied a pure big-bang whitebox approach to a single application system within a solution portfolio, and relied only on the legacy application subject matter experts. The legacy application system was developed in COBOL and the target system in Java. Following modernization, as shown in 3, the cost per service unit of the application reduced from $100 * C_1$, to $139.6 * \frac{C_1}{8} + 44.2 * C_1$, or $61.65 * C_1$. While modernization provided cost savings to the sponsoring users, it also increased the total system utilization to 83.8%, causing concerns for the infrastructure management team. One of the key reasons for the increased utilization was the legacy application developers were unfamiliar with the target language and introduced overhead into the transformed application that could have been avoided with sufficient expertise in the target environment.

These baseline results are consistent with prior efforts within the organization in which only legacy subject matter experts participated in the modernization project.

D. Results

By applying our framework for modernization and utilizing legacy subject matter experts as advisers to target subject matter experts during solution design, the organization was able to achieve far better results than our baseline. As shown in Figure 3, the total system utilization of the modernized application systems remained the same, and the cost per service unit reduced from $100.0 * C_1$ to $21.25 * C_1$.

The simplicity of the application was also greatly increased. The target implementation contains 88% fewer lines of code. It is estimated that there is a 66% reduction in the time required for on-boarding new service consumers[1].

The solution portfolio contains native vendor applications which are ineligible for modernization to the target environment. After modernization, we achieved a 99% reduction in the operational complexity of the solution with approximately 50 batch jobs.

V. Threats to Validity

In this section we discuss the limitations and threats to validity of our modernization framework.

Human Factor: Resistance to modernization, or "putting yourself out of work", is a prevalent concern, as echoed by the authors in [3] and [4]. In our approach, legacy system developers provide their perspectives during the Pre-Processing Application Insights phase, becoming advisers to the target subject matter experts. Thus, assuaging fears associated with job obsolescence due to modernization.

Return on Investment: Return on investment (ROI) is often used by management as the driving factor to justify funding legacy system modernization. Many aspects of modernization are purely technical decisions, such as language choice, middleware platforms and developer operations. In organizations where technology is funded by the line of business, such as in our case study, it is difficult to justify ROI without clear value added to the sponsoring users. Our framework directly addresses this by demonstrating cost savings to the sponsoring user. Specific targets of modernization, such as business rules management systems, will also provide enhanced capability to the sponsoring user, contributing to near term ROI.

Case Study Validity: Our modernization framework was validated using an industrial case study from a large financial services institution. It can be argued that the case study does not encompass the breadth of organizations using legacy systems. The case study was chosen as the organization operates in a high availability environment where loss of service results in significant financial losses. The organization it required seamless modernization to ensure its customer base was not impacted by the changes. Moreover, the legacy code spanned over 30 years of continued use and is representative of typical legacy systems.

VI. Conclusions and Future Work

The results of our industrial case study, on a large, multinational financial services institution, validates the applicability of our approach in modernizing legacy systems. Our approach of emphasizing the existing human resources and knowledge allowed us to reduce the charges to the sponsoring users by 80%, the total lines of code by 88%, the operational complexity by 99% and provided a 66% reduction in the man-hours required for onboarding new customers.

Our future work will expand on the Post-Processing step of infrastructure migration, specifically migrating to cloud infrastructures. We will also investigate the impact of tooling and developer operations on the productivity of workforce candidates beginning their professional careers. Finally, we will validate the drivers for modernization through survey of industrial practitioners.

[1]Determined via informal survey of the application architect and development team. Formal statistics were not readily available.

References

[1] T Sucharov and P Rice. "The burden of legacy". In: *Online: http://www.ncc.co.uk/article* (2005).

[2] GR Gangadharan et al. "IT Innovation Squeeze: Propositions and a Methodology for Deciding to Continue or Decommission Legacy Systems". In: *Grand Successes and Failures in IT. Public and Private Sectors*. Springer, 2013, pp. 481–494.

[3] Belfrit Batlajery et al. "Industrial perception of legacy software system and their modernization". In: *Technical Report Series* UU-CS-2014-004 (2014).

[4] Ravi Khadka et al. "How do professionals perceive legacy systems and software modernization?" In: *Proceedings of the 36th International Conference on Software Engineering*. ACM. 2014, pp. 36–47.

[5] Jesús Bisbal et al. "Legacy information systems: Issues and directions". In: *IEEE software* 5 (1999), pp. 103–111.

[6] K. Lewotsky. "The Pieces Are Falling Into Place". In: *IBM Systems Magazine Digital Edition* July (2015).

[7] Keith Bennett. "Legacy systems: coping with stress". In: *Software, IEEE* 12.1 (1995), pp. 19–23.

[8] Belfrit Victor. "Revisiting legacy systems and legacy modernization from the industrial perspective". In: (2013).

[9] T Rodden et al. "Social viewpoints on legacy systems". In: *Systems Engineering for Business Process Change* (2000), pp. 151–163.

[10] Jesús Bisbal et al. "A survey of research into legacy system migration". In: *Technique report* (1997).

[11] Michael L Brodie and Michael Stonebraker. *Migrating legacy systems: gateways, interfaces & the incremental approach*. Morgan Kaufmann Publishers Inc., 1995.

[12] Santiago Comella-Dorda et al. "A survey of black-box modernization approaches for information systems". In: *Software Maintenance, 2000. Proceedings. International Conference on*. IEEE. 2000, pp. 173–183.

[13] Nelson H Weiderman et al. *Approaches to Legacy System Evolution.* Tech. rep. DTIC Document, 1997.

[14] Narsim Ganti and William Brayman. *The transition of legacy systems to a distributed architecture*. Wiley-QED Publishing, 1995.

[15] Michael L Brodie and Michael Stonebraker. "DARWIN: On the incremental migration of legacy information systems". In: *Distributed Object Computing Group, Technical Report TR-0222-10-92-165, GTE Labs Inc* (1993).

[16] Jane Ransom, I Somerville, and Ian Warren. "A method for assessing legacy systems for evolution". In: *Software Maintenance and Reengineering, 1998. Proceedings of the Second Euromicro Conference on*. IEEE. 1998, pp. 128–134.

[17] Ian Warren. *The renaissance of legacy systems: method support for software-system evolution*. Springer Science & Business Media, 2012.

[18] Bing Wu et al. "The butterfly methodology: A gateway-free approach for migrating legacy information systems". In: *Engineering of Complex Computer Systems, 1997. Proceedings., Third IEEE International Conference on*. IEEE. 1997, pp. 200–205.

[19] Santiago Comella-Dorda et al. *A survey of legacy system modernization approaches*. Tech. rep. DTIC Document, 2000.

[20] William M Ulrich. *Legacy systems: transformation strategies*. Prentice Hall Englewood Cliffs, 2002.

[21] Boby George and Laurie Williams. "An initial investigation of test driven development in industry". In: *Proceedings of the 2003 ACM symposium on Applied computing*. ACM. 2003, pp. 1135–1139.