Webology (ISSN: 1735-188X)
Volume 13, Number 1, 2016

Role Of Legacy Systems In Evolution Towards Service-Oriented

Sajeeda Parveen Shaik[1] , Dr.Y.P. Singh[2] ,
Dr.Akash Saxena[3]

[1]Research Scholar, Sunrise University, Rajasthan.

[2]Supervisor, Sunrise University, Rajasthan.

[3]Co-Supervisor, Sunrise University, Rajasthan.

ABSTRACT

Legacy system software generally comprises a database (sometimes in the form of a set of files) and a collection of application programs in strong interaction with the former. They constitute critical assets in most enterprises, since they support business activities in all production and management domains. Legacy systems: they typically are one or more decade old, they are very large, heterogeneous and highly complex.

INTRODUCTION

Service-oriented architecture (SOA) can be viewed as an architectural construct for flexible connection of separate components in response to changes in business. SOA focuses on the exchange of information among major software components and on the reusability of the components by separating the interface from the internal implementation. There are several features of SOA that make legacy system modernization appealing in today's world, including loose coupling, abstraction of underlying logic, agility, flexibility, reusability, autonomy, statelessness, discoverability and reduced costs. The primary purpose for the adoption of SOA is to improve business communication so that the goals of the enterprise can be more readily realized.

Although Service-Oriented Architecture (SOA) has become popular in recent years, the majority of legacy systems are still not SOA enabled. The increase in the amount of information that companies must handle has resulted in a considerable increase in the complexity of the legacy systems that store this information. While moving to a service oriented architecture platform can help in handling this increase, at the same time it is important to preserve the investment of many years of tuning and debugging of the legacy assets as much as possible.

ADVANTAGES

- Improving reliability,
- Reducing hardware costs,
- Leveraging existing development skills
- Moving to a standards-based server and applications
- Reduced management costs etc.

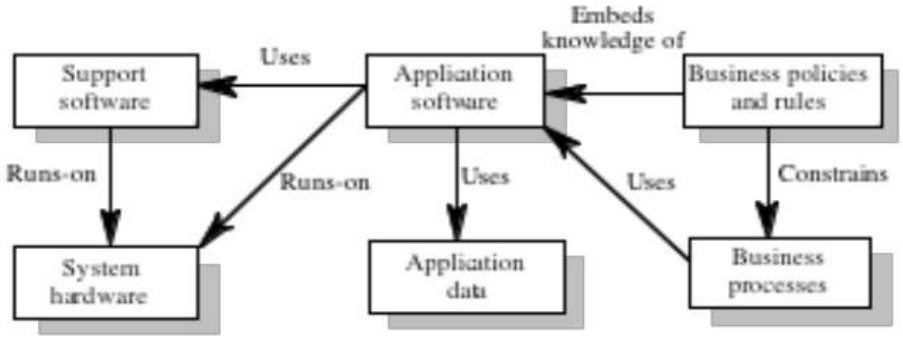

Fig.1 Components of Legacy System

STRUCTURE OF LEGACY SYSTEM

Legacy systems are not simply old software systems although the software components of these systems are the main focus of this study. Legacy systems are socio-technical computer-based systems so they include software, hardware, data and business processes. Changes to one part of the system inevitably involve further changes to other components. Decisions about these systems are not always governed by objective engineering criteria but are affected by broader organizational strategies and politics.

LEGACY SYSTEM DESIGN

Virtually today's entire legacy systems were designed before object-oriented development was widely used in industry. Rather than being organised as a set of interacting objects, the programs in these systems are usually structured as a collection of subroutines or functions. Each subroutine provides part of the functionality of the system and is called, as required, by other subroutines. In some languages, subroutines have their own private data but also access shared data areas. In other languages, such as older versions of COBOL, data is shared and accessible by all subroutines. A function-oriented design strategy relies on decomposing a program into a set of interacting functions or subroutines with a centralised system state shared by these functions (Figure 3.8). The local state information in functions is only maintained while they are in execution. This design strategy is embedded as 'top-down design' or 'structured design' in a number of structured

methods which were invented in the 1970s and early 1980s. Hundreds of thousands of application programs have been developed using these methods and associated CASE tools.

ASSESSMENT LEGACY SYSTEM

Organizations which depend on many legacy systems and which have a limited budget for maintaining and upgrading these systems have to decide how to get the best return on their investment. This means that they should make a realistic assessment of their legacy systems and then decide on what is the most appropriate strategy for evolving these systems. There are 4 strategic options:

1. Scrap the system completely: This option should be chosen when the system is not making an effective contribution to business processes. This occurs when business processes have changed since the system was installed and they are no longer completely dependent on the system. This situation is most common when mainframe terminals have been replaced by PCs and off-the-shelf software on these machines has been adapted to provide the computer support that the business process needs.

2. Continue maintaining the system: This option should be chosen when the system is still required but where it is fairly stable and users do not request a large number of system changes.

3. Transform the system in some way to improve its maintainability: This option should be chosen when the system quality has been degraded by regular change and where regular change to the system is still required.

4. Replace the system with a new system: This option should be chosen when other factors such as new hardware mean that the old system cannot continue in operation or where off-the-shelf systems are available which allow the new system to be developed at a reasonable cost.

KEY POINTS

- A legacy system is an old system that still provides essential business services.
- The quality of the system depends on the quality of the business processes, the quality of the application software itself and the quality of the hardware and software which is used to support the system.
- Most legacy systems have been designed from a functional perspective and are composed of sets of interacting functions which communicate through parameters and global shared data areas.
- Legacy systems are not just application software systems. They are socio-technical, computer-based systems so include business processes, application software, support software and system hardware.

REFERENCES

1. Agrawal, H., 1994. On slicing programs with jump statements In: PLDI ’94: Proceedings of the ACM SIGPLAN 1994 conference on Programming language design and implementation. ACM Press, New York, NY, USA, pp. 302–312.

2. Batini, C., Lenzerini, M. &Navathe, S. B. (1986). A Comparative Analysis of Methodologies for Database Schema Integration.ACM Computing Surveys 18 (4), 323-64

3. Elmasri, R &Navathe, S. B. (1984). Object Integration in Database Design. Proceedings of IEEE Conference on Data Engineering Los Angeles

4. Elmasri, R. &Navathe, S. B. (1994). Fundamentals of Database Systems Benjamin/Cummings Publishing

5. Emmrich, W, Ellmer, E. &Fieglein, H. TIGRA: An architectural style for enterprise application integration. Proceedings.23 rd International Conference on software engineering (ICSE-01), 567-76