Webology (ISSN: 1735-188X)
Volume 13, Number 2, 2016

Reviewed Study On Legacy System And Data Base Management

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 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. Many of them significantly resist modifications and change due to the use of ageing technologies and to inflexible architectures. Since they nevertheless are due to evolve, sophisticated techniques have been elaborated that allow programmers and developers to identify and understand the logic of the code fragments and of the data structures that are to be changed, despite the lack of precise and up to date documentation. Recovering the required knowledge and control of poorly documented software components is the main goal of software reverse engineering.

INTRODUCTION

System development largely ignores the database component. The latter appears as an encapsulated subsystem acting as a reliable and efficient data server. The database is an appropriate data container that ensures persistence, limited consistency, smooth concurrency and accident resistance. When external data are necessary, the programs invoke data extraction services from the DBMS through some sort of data manipulation language (DML). The same channel is used to send data to be stored in the database. The emphasis is less on the domain modeling aspect of the database than on convenience (such as storage transparency) and performance.

According to the modern description of pure software systems, a typical database can be perceived as a software sub-system made up of several thousands of classes, comprising dozens of thousands of attributes, and connected through several thousands of inter-class associations. Each class can collect several millions of persistent instances, so that a typical database forms a semantic network

comprising billions of nodes and edges. These instances are shared, possibly simultaneously, by thousands of

programs that read, create, delete and update several thousand times per second. Any of these programs can include hundreds of database statements of arbitrary complexity.

Since the database is supposed to include all the pertinent data about all the static object types of the application domain and since each program is designed to translate a business activity relying on these objects, it should not come as a surprise that data and processing aspects are tightly intertwined in application programs.

Reengineering, also known as renovation, is the examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form. Reengineering generally includes some form of reverse engineering (to achieve a more abstract description) followed by some more form of forward engineering or restructuring. Migration is a variant of reengineering in which transformation is driven by a major technology change.

Chikofsky (1996) defined data reverse engineering as "a collection of methods and tools to help an organization determine the structure, function, and meaning of its data". Database Reverse Engineering (DBRE) is the process through which the logical and conceptual schemas of a legacy database, or of a set of files, are reconstructed from various information sources such as DDL code, data dictionary contents, database contents or the source code of applications that use the database. (Hainaut et al., 2009).

Any process that consists in deriving artifacts from other artifacts relies on such techniques as renaming, translating, restructuring, replacing, refining and abstracting, which basically are transformations. Most database engineering processes can be formalized as chains of elementary schema and data transformations that preserve some of their aspects, such as its information contents (Hainaut, 2006). Information system evolution, and more particularly system migration as defined in this thesis, consists of the transformation of the database and of its programs into a new system comprising the modified database and the adapted programs. As far as programs are concerned, the transformations must preserve the behavior of the interface with the database management system, although both the syntax of this interface and/or the underlying data structure may undergo some changes.

The history of an engineering process is the formal trace of the transformations that were carried out during its execution. Each transformation is entirely specified by its signature. The sequence of these signatures reflects the order in which the transformations were carried out. The history of a process provides the basis for such operations as undoing and replaying parts of the process. It also supports the traceability of the source and target artifacts.

A General Overview of Database Migration:

Legacy information system migration is a major research and business issue; few comprehensive approaches to migration have been developed. Given the bewildering array of legacy systems in operation and the problems that they pose it seems unlikely that a generic migration method suitable for all legacy systems is possible. However, a set of comprehensive guidelines to guide migration is essential. Before embarking on a migration project, an intensive study needs to be undertaken to find the most appropriate approach of solving the problems the legacy system poses. To the best of authors' knowledge, no successful, practical experiences of real migration projects, following a comprehensive migration approach have yet been reported. The few (successful) migration-like reports found in the literature ([Ocal96][Aebi97]) describe ad-hoc solutions to the problem at hand.

Although the vast majority of literature included in this review is relatively recent, I also include several articles from earlier years which provide useful insights into the early stages of data migration. The research of Housel et al.'s 1974study at IBM is one such example where some of the main issues in data migration are summarized and future research areas are suggested. Their stated principal reasons for data migration have remained true and are as applicable today - at least at a general level - as they were 30 years earlier. These reasons include

1) A change in the hardware system,

2) Conversion from one system to another,

3) A change in the structure or program as a result of modified application requirements, and

4) The addition of an application to an existing database. Other significant early contributions include Fry (1970) and McGee (1970), who suggest the creation of a data and storage structure definition language.

Alternatively, Young (1970) proposes a procedural data structure mapping technique whereas Sibley and Taylor (1970) suggest a similar technique, but propose using a nonprocedural approach. Another important contribution was a PhD dissertation by Smith (1971) who began to address generalized issues of data translation. A common feature of the 1970s research is a focus on the definition of common languages for the purpose of defining data, storage, and mapping processes.

Migration between Different Database Models:

Another important research area is the consideration of migration from relational to object-oriented databases. Monk et al. provide a sound foundation in their 1996 article on the topic. Not all environments are suitable for this type of migration, since many migrations remain relational or object-oriented rather than changing from one system to the other. In cases where the programming

model changes, however, there are two main approaches. First, one can consider implementing a layer of object-orientation on top of the relational database backend. Crowe (1993) along with Hardwick and Spooner (1989) provide instances where this kind of approach can function effectively. The IRIS system is another such example which is described by Wilkinson et al. (1990) and Fishman (1987). In this example, an object-oriented DBMS is developed on top of an existing relational DBMS. The advantage of such an approach is that the relational data is still accessible as relational data; the disadvantage is the inefficiency of having to translate data manipulation language (DML) commands between the two layers. The second approach is to implement more of a migration rather than simply to overlay an interface.

In this case, relational technology is migrated to objects (Monk, 1990). The most significant step in this process is to derive an object-oriented scheme from a relational scheme from the existing source system. Chiang (1994), Hainaut (1991) and Premerlani et al. (1994) have researched this area of reverse engineering relational databases to extract an ER model for evolutionary purposes. The natural extension of this research is to transform the ER (or EER) model into an object-oriented schema.

Impacts of Legacy System on Business:

In the introduction to this study, it was noted that legacy systems are simultaneously business assets and business liabilities.

1 As a Assets: They are assets in that, through years of debugging effort they have grown to reflect essential tacit knowledge - underpinning many of the organization’s business processes. It is this hard-won reflection of tacit knowledge – in terms of both action potential and action memory - that constitutes the business asset. The real, and justified, fear of organizations is that a new information system replacing a legacy system may well accurately support the knowledge, of which a business is explicitly aware and dependent.

2 As Liabilities: They are liabilities, however, in that it has become increasingly difficult to adapt them to reflect ongoing business process change [3]. Since an information system is, for the most part, designed with its initial requirements in mind, it is no surprise that the designs - the internal structure or architecture of mature computer systems are typically inappropriate for the substantially changed requirements they face in later life. As contract negotiations and hence requirements evolve, mature information systems tend to be 'hacked' to force their aging designs to reflect these changes. Thus, over time, they grow old disgracefully. This results in what we term accidental architecture - the architecture of the information system grows, for all intents and purposes, accidentally as a consequence of repeated waves of hacking - emerging more from the ingenuity of the systems developers' ability to 'warp' the code than from any purposeful design. As successive waves of hacking warp the information system further and further, its accidental architecture becomes more and more convoluted, and the intellectual capability of the developers

to identify and apply clever hacks diminishes to a point at which the system is declared un maintainable. That is, the legacy system grows disgracefully, until it reaches a point where it resists reflection of further business process change. Consequently, a legacy system fails to support the evolving processes of a business – either forcing the business to work around rather than with the information system or even to abandon essential process changes altogether. At this point the system has become a business liability.

Importance of Assessment of Legacy System:

To support a living, learning, organization successfully, we must maintain the reflection of tacit knowledge which makes legacy systems into assets, whilst simultaneously striving to eliminate the resistance to reflecting business process change which makes those same systems into liabilities.

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.

Approaches for Legacy System Evolution:

Several approaches have been proposed to the problems faced in evolution of Legacy Systems. We divide the various approaches into three general categories.

- Redevelopment
- Wrapping
- Migration

Conclusion

Businesses which have a large number of legacy systems are therefore faced with a fundamental dilemma. If they continue using the legacy systems and making changes as required, their costs will inevitably increase. If they decide to replace their legacy systems with new systems, this will be costly and the new systems may not provide as effective business support as the legacy systems.

Hence this paper describes the impact and importance of legacy systems in business different approaches of Legacy System Evolution so that an organization can gain benefits from its Legacy data.

A Legacy system is what some refer to as “yesterday’s system”. In the context of computing, legacy refers to outdated computer systems, programming languages or application software that continues to be used, typically because it still functions for the user’s needs, even though newer technology or more efficient methods of performing the task are now available. A legacy system may include procedures or terminology which are no longer relevant in the current context, and may hinder or confuse understanding of the methods or technologies used.

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. Clarinval, A., 1981. Comprendre, Connatreet Matriser le Cobol, 2nd Edition. Presses Universities de Namur
4. Dumpala, S. & Arora, A. Schema translation using the entity-relationship approach. In S.T. March (Ed.), Entity-Relationship Approach (pp 337-56), Amsterdam: North Holland.
5. Elmasri, R &Navathe, S. B. (1984). Object Integration in Database Design. Proceedings of IEEE Conference on Data Engineering. Los Angeles.
6. Hainaut, J.-L., Henrard, J., Englebert, V., Roland, D., Hick, J.-M., 2009. Database Reverse Engineering. Springer, pp. 263–263, to appear.
7. Johanneson, P. (1994). Linguistic Instruments and Qualitative Reasoning for Schema Integration.Proceedings of the third international conference on Information and knowledge management. (pp. 252-62). New York: ACM Press.
8. Kelly, C. &Nelms, C. Roadmap to checking data migration. Computers & Security 22 (6) 506-510.
9. Munir, K., M. Hassan, W., Ali, A, McClatchey, R. &Willers, I. (2002). Database independent migration of objects into an object-relational database. Proceedings. 2nd International Workshop on Autonomous Decentralized System (pp. 132-39). Beijing, China.