Multidimensional Legacy Aspects of Modernizing Web Based Systems

Henryk Krawczyk[1], Konrad Dusza[1], Łukasz Budnik[1], Łukasz Byczkowski[1]

[1] Gdansk University of Technology,
Faculty of Electronics, Telecommunications and Informatics
ul. Gabriela Narutowicza 11/12, 80-952 Gdańsk, Poland

Abstract. The paper presents basic legacy transition techniques used in software lifecycle either on system or component levels. It discusses a user case of the Endoscopy Recommender System. It also considers an impact of requirements, programming platforms, software development strategies and software standards on legacy status of web applications.

1 Introduction

With web technologies developing at a growing pace and IT systems being adopted into business models, there occur situations where increasing number of companies face the urging need for serious changes in their IT systems [1]. Legacy Information Systems can be defined as “any IT system that significantly resists modification and evolution” [2] that most often are the IT backbone of a company [3]. In general, the topic of Legacy Information Systems has been thoroughly examined and the common problems are well identified [3][4][5]. Even though, case studies show that no miracle cure for “migration migraine” has been developed. However, there are three main concepts regarding coping with Legacy Information Systems. These are [3] (from the most lightweight, to the most revolutionary one) as follows:

Wrapping – accomplished by developing a small software component that connects a legacy component with a new component. A wrapper serves as a translator in communication between these components.

Migration – a much more complex approach, used when both wrapping and redevelopment cannot be afforded either in terms of risk level or when transition between components must be transparent.

Re-development – means developing a component from scratch, usually reimplementing a component in a different programming language.

All these strategies can be deployed on both the system level and on the component level. For instance, we could wrap the business tier, re-develop presentation tier and migrate data tier in a web application.

The next section discusses attributes that describe legacy characteristics of web-based legacy IT systems. Section 3 describes a solution for legacy problems supported by a real-world case study. It concerns modernizing Endoscopy Recommender

System working at the Medical University of Gdańsk. Besides, it provides a general methodology of developing legacy transition strategies in web based IT systems. The concluding section presents general suggestions relating to inclusion of legacy factors into software life-cycles.

2 Legacy Attributes of Web Applications

One can define a typical web-based legacy information application as a system which functions are crucial for supporting business in the company of which upgrade involves a high degree of risk. One example is an e-shop based on Apache 1.3, Perl CGI-scripts and MySQL 3.2 DBMS in which business logic and presentation layer are intertwined. The main questions concerning legacy issues are: When does a system become a legacy one? When a transition cannot be avoided? A legacy system is a system that still fulfills software requirements imposed by the contract under which the system has been developed.

Below is the analysis of engineering-related external factors that are the catalysts of changes in the system. They can be divided into four main categories: extra functional requirements and expectations, technological platform changes, software architecture modifications, standards and interoperability support. These categories of changes are illustrated in Fig.1 – Fig. 4.

The most common reason for serious improvements in an information system are changes in software requirements. Apart from functional ones, there are groups of other requirements that are often not fulfilled in the first version of a system. The increasing importance of different kinds of requirements in time is suggested in Fig.1.

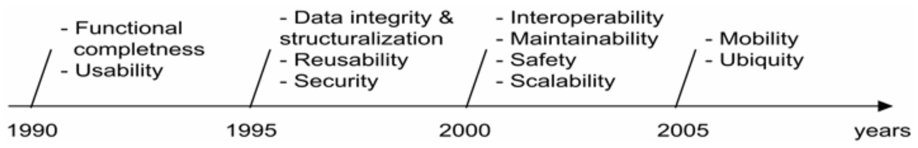

Fig. 1. Distribution of requirements importance in web based systems.

Today, there is a plethora of technologies that support web application development (see Fig.2). Decision on rewriting a web application to a different technological platform can be taken to: achieve better scalability, efficiency and maintainability; show customers that the company uses the newest technologies; merge with other systems written in other technologies. Some migrations are easy to conduct from that point of view, eg. from PHP3 to PHP5, others require quite an effort (eg. PHP to J2EE). In most cases, change of technological platform by itself should not be the only reason to conduct migration.

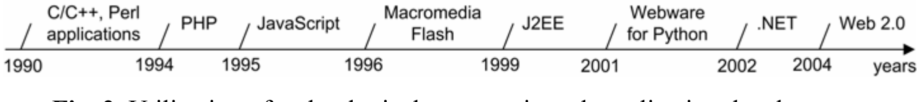

Fig. 2. Utilization of technological progress in web application development.

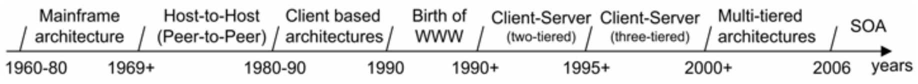
Mainframe architecture
Host-to-Host (Peer-to-Peer)
Client based architectures
Birth of WWW
Client-Server (two-tiered)
Client-Server (three-tiered)
Multi-tiered architectures
SOA
1960-80
1969+
1980-90
1990
1990+
1995+
2000+
2006
years

Fig. 3. Trends in distributed software architecture development.

Switching between different technological platforms is often accompanied by a decision to improve system's architecture during migration (see Fig.3).

Software architecture evolves towards multi-tiered applications and SOA [7], which are meant to be the tools for achieving business flexibility in on-demand solutions. However, real-world web-based legacy systems often have data, business and presentation intertwined, which is often a result of inappropriate development process and setting aside the principles of software design for the sake of approaching deadlines.

During migration process, we might want to use a different software engineering methodology than the one used during development of a legacy system. The will to reconstruct the system in a different way is rarely the sole reason to migrate. Similar situation arises with quality management. Migrations are often occasions to introduce quality management into the software development process. In general quality management can contribute to the fact that the system will not be considered as a legacy one for a long time.

Web application environments also include a numerous group of quickly evolving standards that the application should comply with (see Fig.4). Introducing new functionalities into application often involves conforming to a certain web standard, eg. news headlines in RSS. When some web standards supersede others a web application that does not conform to new standards is often considered legacy. However, in most cases, wrappers should be a sufficient solution for such problems.

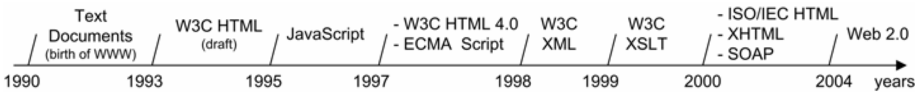

Fig. 4. Evolution of standards for web applications.

When determining the legacy status of an application, we should firstly determine its current position on each of these timelines. The distance between the present date and the latest date corresponding to the desired state of the system is a measure for the system's legacy level, which will help to determine the need for software development. If a difference can be seen only in one or two aspects, then perhaps a simple transition should be reconsidered. If not, developing a complex, component-level transition strategy is recommended, with carefully planned use of wrapping, migration and re-development techniques.

Another legacy aspect is system interoperability, meaning, that if maintenance phase changes distort system's communication with other systems, then the other systems become legacy ones. Such situation is usually unacceptable for most of primary system's users. For example, our e-shop cooperates with another e-shop, which was forced to change technology platform from ASP.NET to J2EE. Previously, we used .NET remoting to access another shop's data, now we have to switch to either

Web Services or Java RMI. In such a case, we have to create new communication module or wrap an existing one and introduce it into our system. The only way to avoid or at least postpone interoperability-driven transitions is to develop systems with high flexibility and extendibility. However, it is a very difficult task in practice.

3 A Solution for Legacy Problems with ERS Example

Transformation types discussed in Section 2, were applied in ERS development as shown in Table 1. The first implementation of the Endoscopy Recommender System (ERS) was deployed in 1997 as a standalone application, without using web technologies. The next generations of the system were introduced in 2001 and 2005 respectively. Table 1 presents a detailed history of the system along with technologies employed in each generation of the system and theirs key features.

Table 1. History of ERS development.

Version, release date	Features	Used technologies	Legacy transformation approach
ERS 1, 1997	Database of patients and examination data. Reports and statistics generation.	MS-DOS, Clipper	none (first ERS version)
ERS 2, 2001	MST standardization of examination descriptions, client-server architecture, replication of medical data for reliability improvement.	Windows and Linux, PHP4, MySQL 3.23, Java 1.2, Apache 1.2	re-development (transition to web based platform)
ERS 3, 2005	New, three-tiered architecture, DVD medical data analysis, security, safety and data integrity assurance. Addition of new reports and other functions requested by client.	Windows and Linux, PHP5, MySQL 5, Apache 2, XML, XSLT, SOAP	DBMS communication wrapping, inner-system data-flow migration to XML, presentation and business tier re-development

As shown in Table 1, the system became legacy two times, in 2001 and 2005. The reasons for transitions were as follows:

1. System requirements were defined incrementally because of extra needs of system users – physicians.
2. Personnel rotation in system development team of successive versions (always students of our faculty, each time with better knowledge of new software technology)
3. Emergence of new web technologies provided means for achieving better implementations of functionalities and higher quality.

Below, we focus on transition from ERS 2001 to ERS 2005. Among different approaches we have decided to use the following one:

1. Begin with architectural changes.

2. Switch to a new technology (writing new source code).
3. Implement new functionalities.

This approach allows us to transform the system into a three-tiered application in natural way. Medical environment is very volatile, which urged ERS to be highly flexible and adaptive. Its component architecture was developed mainly to fulfill that need. Development of a new, properly tiered architecture enabled designing a system engine based on XML and XSLT processing. Transition to the latest PHP, MySQL and Apache versions available made it possible to implement a broader set of requirements.

The ERS database engine was migrated from MySQL3 to the newest MySQL5 and took full advantage of its new DBMS features (see Table 1). All broken interrelations were copied into separate database. Medical data collected by many years should never be deleted or discarded. Instead, they should be stored in safe and secure archives – this data is a great source of information. Foreign key constraints were added – responsibility for foreign key checks was transferred from programmers to DBMS and is processed automatically without any interference. New ERS uses also trigger mechanism for consistency checks during e.g. delete operations. Data storage engine was moved from MyISAM to InnoDB. Transaction support was implemented in the target system and the new ERS now works in fully transactional mode.

To cope with intertwined business logic tier, our team created a template of the ERS business logic, which proved very useful in further development. Previously, business logic of the system was highly integrated with other tiers, which forced programmers to carefully analyze this aspect and separate respective business functions. As a result, an XML file was created, which contained data describing division of business functionalities into modules and structure of entire system. Later we used the XML business logic files to automatically generate directory tree, database queries and even code templates for the system, which was achieved by building different sets of XSL transformation sheets. XML business logic files also helped developers to keep references between parts being re-developed and corresponding functionalities in the legacy version of ERS. We found this feature particularly useful when assuring that the new version meets all functional requirements that the old system met.

Although the ERS interface turned to be proper for managing functionalities offered by the system, and it did not need updating itself, a new system architecture forced developers to isolate presentation logic from the rest of a system, which was highly interspersed with business and data tier's code in the previous ERS generation.

Knowledge collected during earlier stages of ERS development helped to decide, which parts of the system can be transformed and how. Analyses of risk, costs and benefits have shown that the structure of legacy data in the system should remain unchanged. What could have been done was the creation of wrappers enabling accessing previous, legacy-structured data-tier in order to migrate to new MySQL5 DBMS with all the latest transaction techniques. The rest of the system was re-developed in correlation with new ERS system architecture. It guaranteed easy modifications and expansion, which is highly valued in system user's environment.

Our experience gained during the development of ERS 2005 shows that a methodology to create transition strategy can be developed and included in a software lifecycle development. A system becomes legacy one during maintenance phase, when system's current features no longer satisfy the needs of users and its environment. Moreover, the legacy state is periodic, and should be expected in every life-cycle regardless of software engineering techniques and technological frameworks used. Web applications are even more endangered to legacy issues as the technologies used in this area of IT are not mature and evolve faster than in other areas.

4 Conclusions

Software life-cycles foresee the needs for smaller changes of software and its requirements during different phases and at the same time neglect the legacy issue caused by both user requirements changes and technological progress. The legacy boundary is often flexible and the legacy state is proclaimed arbitrarily by business-related managers regardless of the life-cycle.

In order to conduct a transition from a legacy system to a newly developed one, one of the three approaches can be adopted both on system and a component level. These approaches are wrapping, migration and re-development. They differ in terms of software re-use that can be applied and the effort that has to be committed to the transition process.

Our work on ERS has shown that further legacy transitions of information systems are inevitable. However, the integration of the legacy state into our software life-cycle should reduce the cost of future legacy transition, due to greater flexibility of a system architecture and design. In this case, previous transition took 18 months, and the present one – 15 months, measuring from the decision to initiate legacy transition to deployment of the final product.

References

1. Flawn D, The Legacy Systems Dilemma Fujitsu, Legacy Migration, http://www2.cio.com/consultant/report2337.html
2. Brodie M., Stonebraker M., Migrating Legacy Systems: Gateways, Interfaces and the Incremental Approach, Morgan Kaufmann Publishers, Inc. USA, 1995
3. Bisbal J., Lawless D., Wu B., Grimson J., Legacy Information System Migration: A Brief Review of Problems, Solutions and Research Issues, Computer Science Department, Trinity College, Dublin, Ireland 1999
4. Hassan A. E., Holt R. C. A Lightweight Approach for Migrating Web Frameworks, Software Architecture Group (SWAG), Department of Computer Science University of Waterloo, Waterloo, Canada 2004
5. Hassan A. E., Holt R. C. A Visual Architectural Approach to Maintaining Web Applications, Software Architecture Group (SWAG), Department of Computer Science University of Waterloo, Waterloo, Canada 2002
6. Krawczyk H., Knopa R., Kruk S., Mazurkiewicz A., Zieliński J., Predictive-incremental strategy of application development, KKIO 2001, Otwock, Poland
7. OASIS, Reference Model for Service Oriented Architecture, http:// www.oasis-open.org/committees/download.php/16628/wd-soa-rm-pr1.pdf