Legacy Systems Assessment to Support Decision Making

Aniello Cimitile*, Anna Rita Fasolino**, Filippo Lanubile***

* University of Salerno, Faculty of Engineering in Benevento, Italy, email: cimitile@unina.it
** University of Naples “Federico II”, Italy, email: fasolino@unina.it
*** University of Bari, Italy, email: lanubile@cs.umd.edu

Keywords: legacy systems, software assessment, construct validity, business value, obsolescence, decomposability, deterioration

1. Introduction

Making decisions about the destiny of the software portfolio is today one dominant concern for those business organizations that own legacy systems. There is a number of options available in managing legacy systems. Typical solutions include: discarding the system and building a replacement one; freezing the system and using it as a component of a new larger system; modifying the system to give it another lease of life. Modifications may range from a simplification of the system (reduction of size and complexity) to ordinary preventive maintenance (re-documentation, restructuring and reengineering) or even to an extraordinary process of adaptive maintenance (interface modification, wrapping and migration). These possibilities are not alternative to each other but making decisions on which approach, or combination of approaches, is most suitable for any particular legacy system are usually taken on the basis of conventional wisdom. Like any decision process, it requires that critical information depicting the state of subject systems is available to the management, to make valid decisions. In these cases, both technical and economic aspects of subject systems must be assessed in order to justifying each decision. Sneed [Sne96], for instance, proposes an approach for planning the reengineering of legacy systems through a five-step process, starting with an analysis of software and ending with contract negotiation. The process includes portfolio analysis for classifying applications according to their technical quality and business value, and for approaching their reengineering accordingly.

A more general approach is now being proposed in Italy for managing and maintaining legacy information systems of the Italian Public Administration. A preliminary assessment of the status (size and main characteristics) of the software portfolio of government organizations highlighted that the wider part of this software (consisting of about 230,000 KLOC) presents all the typical symptoms of legacy code. Maintaining these systems consumed about 78% of global investment in information technology made by government organizations in 1995. In order to gain cost reduction and process optimization, AIPA[1] suggested that the improvement of the maintenance productivity could be achieved by promoting preventive maintenance. It also stressed the need for defining, within the single government organizations, adequate projects for simplifying, re-documenting/ re-engineering the running systems, and migrating them toward modern platforms.

To assist government organizations to fulfill these strategic objectives, AIPA devised the need for a reference life cycle for legacy systems to be used as a standard decision framework for the specific system. As a result of a research project jointly carried out by AIPA and the University of Salerno, Faculty of Engineering in Benevento, a first draft of this reference model has been produced [Canf97]. Decision points in the model should be based on the ability to assess both the technical and the economical aspects of a legacy system. The technical aspects include system and software features that are relevant from a maintainer point of view, such as the obsolescence of the system, its maintenance degradation, the decomposability of its architecture. The economical aspects include characteristics that are relevant from a management point of view, such as the strategic role that a legacy system plays to fulfill the goals of the business organization.

2. Problem being addressed

The goal of our research project is to support the decision-making process behind the reference life cycle presented in [Canf97] through a systematic assessment of legacy systems during their evolution.

The basic idea of the above reference life-cycle model is that all systems undergo a continuous evolution by switching among four main phases: *simplification*, *ordinary reactive and preventive maintenance*, *extraordinary adaptive maintenance*, and *replacement*.

[1] AIPA is the national control organisation established in Italy in 1993 with the aim of improving the delivery of public services through the best use of information technology.

The *simplification* phase consists of reducing the size of the system to be maintained by eliminating dead code, and removing unused functions and data. In the legacy systems from Italian government organizations, the portion of unused functions and data can be surprisingly high essentially due to practices of quick-fix maintenance and business processes no more supported.

The phase of *ordinary reactive and preventive maintenance* covers the maintenance activities defined by the IEEE standard 1219 [IEEE93]: emergency repair, corrective maintenance, adaptive maintenance, perfective maintenance and preventive maintenance. In Italy, government organizations give these activities on contract to outsourcing companies. The cost of the contract is usually proportional to the size of the system, and this is the reason because the simplification phase should be identified as a different phase performed by organizations other than that under contract for ordinary maintenance.

The phase of *extraordinary adaptive maintenance* is triggered by major changes in the business processes and the way users perceive those. In the domain of public administration, major changes are related to administrative processes and how public services are delivered. For example, introducing a GUI interface on a system implemented with character-based displays, and thus changing the underlying metaphor behind a system, may have a dramatic impact on the users both in terms of productivity and satisfaction. Wrapping old components to interface modern hardware and software has become a popular approach to migrate old systems to a client / server architecture.

The phase of *replacement* entails the development of new software to replace the old existing ones based on a global plan for substitution. During the replacement, the existing system is "frozen" and not maintained anymore except for emergency repair.

All these four phases are different with respect to what they deliver as well as to their pre-conditions. The conditions that must be checked, before to jump on any of these phases, are based on four basic attributes of a legacy system: business value, decomposability, obsolescence, and deterioration. The abstract concepts that define these attributes are shown in Table 1.

Attribute	**Definition**
Business Value	It expresses to what extent a software system is essential to the business of an organization
Decomposability	It expresses how easily the main components of a software system are identifiable and independent from each other
Obsolescence	It expresses the aging of a software system caused by the failure to meet changing needs
Deterioration	It expresses the aging of a software system as a result of continuing changes that are made

Table 1. The four attributes to assess a legacy system in the reference life cycle

In order to explain how the four attributes might be used to support decision-making in the reference life cycle, let's suppose we are able to measure them on an ordinal scale with two values ("high" and "low") and that the contract of ordinary maintenance is at the end. Such a system will be submitted again to *ordinary maintenance* if its business value is "high" and obsolescence is "low". Otherwise, if the obsolescence of the system is "high", the decision about the new maintenance action to be entered will depend on the values of the other attributes. If business value and decomposability are "high", while deterioration is "low", the *extraordinary maintenance* phase will be entered. On the contrary, if business value and decomposability are "low", but deterioration is "high", then the system will be replaced (*replacement phase*).

3. Measuring the constructs of interest

To achieve our goal of supporting the decision-making process for a legacy system we need to measure successfully the four basic attributes. Likewise each abstraction that can be defined from real world observation, the four basic attributes of a legacy system are abstract concepts that cannot be directly measured. They require some concrete representations that approximate what is meant when speaking about them. In social science, the abstract concepts (in our case, the basic attributes) are called *constructs* and the concrete representations we are looking for are called *variables* [Judd91]. For any construct, there can be many different ways to measure it, that is more corresponding variables. Once a variable has been identified, the sequence of steps to measure the variable (expressed as a number or a rank) is its *operational definition*. The degree to which the constructs of interest are successfully operationalized is the *construct validity* of a study.

The main problem behind construct validity is that variables never measure only the construct of interest but they also measure other things that are irrelevant. The irrelevant characteristics represent the error of the measurement. The error can be decomposed in a systematic and random error. The *systematic error* reflects the measurement of constructs that are different from that in which we are interested. For example, the systematic error when measuring the user satisfaction by counting positive answers to questionnaires includes things such as asking the wrong questions and low motivation of people. The *random error* reflects nonsystematic factors that can influence the measurement. Using the same example of measuring user satisfaction through questionnaires, the random error includes things such as decline of concentration when answering, and inability to fully understand the questions.

Yin [Yin94] suggests three tactics to maximize the construct validity: triangulation, chain of evidence, and key informants as reviewers. The first tactic, triangulation, is related to measure the construct of interest in different ways and then cross checking the results to if they converge or not. Triangulation can be done with respect to the data sources (documents, source code, archival records, interviews, surveys, direct and participant observation) as well as with respect to the variables (multiple operationalism). The second tactic, maintaining a *chain of evidence*, consists in letting external reviewers to follow a logical chain that link the initial constructs to the final measures. This tactic is usually verified when a study goes under a peer-review process at conferences and journals. The third tactic, the use of key informants as reviewers, consists in the discussion of intermediate and final results of the study with people from whom information has been gathered, and in the elicitation of their comments ("do you agree with these measures?").

3.1 Measuring Business Value

The *business value* expresses to what extent a software system is essential to the business of an organization. In the case of the domain of public administration, what is essential is the ability of a software system to achieve efficiency and fairness in the administrative activity. This ability greatly depends on the complexity of the business processes and administrative rules that the system implements. For example, a software application has a critical business value if it encapsulates undocumented knowledge to realize highly specialized domain functions that are needed to accomplish the strategic and institutional mission of the organization. The business value is scarcely critical when the application realizes well-understood and fully documented domain functions. The business value is non-critical when the application realizes useful functions that are nevertheless unrelated to the specific application domain (for example, office automation facilities). The business value is irrelevant when the application is never executed either because of structural reasons (dead code) or because it realizes functions which are not (or no more) pertinent to the organization.

- *What can be observed and measured?*

The business value of a legacy system might be expressed by (1) its *utility* in the administrative activity: a system that is never employed (no matter why), is useless and thus of irrelevant business value. On the contrary, a legacy system that must be operational all time has critical business value. Another representation of business value may be (2) the *contribution to profit* it produces in the business processes: if it scarcely contributes to the profit of the organization, its business value is scarcely relevant. Another representation of the business value might be (3) the *information relevance*: if data are accessible only through the legacy system then its business value is critical. Finally, the business value might be represented by (4) the *specialization* of the legacy system with respect to the business domain: office-automation functions can be easily substituted with COTS products, whilst high specialized and strategic domain functions cannot.

- *How data could be collected?*

Data sources to be considered for assessing the business value of a legacy system might include (1) users and managers from the business organization; (2) user documentation, like user guides and similar; and (3) descriptions of the persistent data of the application.

The procedures for collecting data might include (1) direct observation, interviews and written questionnaires to be submitted to users in order to assess, for instance, how many persons from the organization use the system's procedures, and how often they do it. In this case sampling strategies should be selected for identifying sub-groups of system functions. Other qualitative methods for collecting data might include (2) the analysis of data documentation in order to assess what is the relevance of data managed by the system; and (3) the analysis of system documentation in order to understand the degree of specialization of implemented domain functions.

3.2 Measuring Decomposability

Decomposability expresses how easily the main components of a software system are identifiable and independent from each other. The ability of building large systems using orthogonal sub-components greatly depends on hiding major design decisions inside modules [Parn72]. An example of orthogonal architecture in the domain of visual interactive software tools is in [Rajl96].

A classification of legacy systems with respect to this attribute is in [Brod95]: decomposable, semidecomposable, nondecomposable, and hybrid. In a decomposable architecture the interfaces, applications, and data management services can be considered as distinct components with well-defined interfaces. In a semi-decomposable architecture only user interfaces and system interfaces are separate modules, while the applications and data management services are not separable. An nondecomposable architecture consists of components that are not separable. Hybrid architectures consist of both decomposable and nondecomposable parts.

- *What can be observed and measured?*

Descriptions of components of the software architecture should be available in order to assess if separation of concerns is applied and eventually classify the components using the scheme in [Brod95]. More variables could be considered by changing the level of granularity (e.g., subsystem, module) and the perspective [Soni95]. The type of representation to be adopted for representing architectural information depends on the level of granularity chosen for the components.

- *How data could be collected?*

The data sources to get an architectural description might be: interviews to original developers and maintainers, design documentation, and source code to be reverse engineered for extracting design documentation.

3.3 Measuring Obsolescence

Obsolescence expresses the aging of a software system caused by the failure to meet changing needs. Obsolescence is the effect of the first Lehman's laws on software evolution [Lehm85] and it corresponds to what Parnas considers as one of the two types of software aging [Parn94]. Continuous progress of hardware/software platforms, programming languages and development practices causes software technology to become outdated in a short time. Obsolescence produces an indirect cost because of not taking advantage of the opportunity to cut maintenance expenses, and more generally to gain position in the business market.

- *What can be observed and measured?*

Here, we assume that an order relationship can be established among the technologies that have been used for developing software systems. Some macro-measures have been proposed recently. Lewis uses computing metaphors as a criterion to rank software systems on an ordinal scale [Lew97]: mathematics, literature, direct manipulation, simulation, and reality. Shaw and Garlan specifically address the evolution of shared information systems in business data processing [Shaw96]: batch processing, interactive processing, unified schemas, and multidatabase. Micro-measures of obsolescence might consider, for example, if the infrastructure technology (hardware, operation system, communication software, DBMSs, GUIs) adopted by a legacy software system is still available on the market, if it is still supported for assistance, and what is its distance from current releases.

- *How data could be collected?*

Data sources to be considered for assessing the obsolescence of a legacy system might include: surveys of technical literature, interviews to expert opinions, inspection of user manuals, software documentation, and source code, and observation of the infrastructure technology itself.

3.4 Measuring Deterioration

Deterioration expresses the aging of a software system as a result of continuing changes that are made. Deterioration is the effect of the second Lehman's law on software evolution [Lehm85] and it corresponds to what Parnas considers as the second type of software aging [Parn94]. Ordinary maintenance are generally performed without respecting what Brooks calls the "conceptual integrity" of a system [Broo95]. Deterioration is also increased by the time constraint that makes maintainers do not update documentation. Typical symptoms of software deterioration include: (1) performance degradation, as a result of program size growth and corrupted structure; (2) reliability degradation, because of new errors introduced as a side-effect of past changes; (3) modifiability degradation, as a result of the fact that degraded code is neither well-understood nor well-documented.

- *What can be observed and measured?*

To asses deterioration, we might consider how a number of affected variables change over time. Examples of variables to be monitored are: error rate, mean time between failures, request change backlog, time to satisfy a request change, time to implement a change once approved, and response time.

- *How data could be collected?*

Data sources to be considered might include: (1) documentation and code, when managed under a software configuration management tool, to compare product metrics across versions, (2) archival records of software problem reports and change requests, and (3) archival records and simulation of performance benchmarks.

4. Parallel or subsequent studies that could be performed

The reference life cycle presented in [Canf97] needs that the basic four attributes (business value, decomposability, obsolescence, and deterioration) have to be successfully measured. We advocate the use of multiple operational definitions, as any single operational definition is imperfect and can be affected by different set of errors or biases. We also intend to follow the other two tactics to maximize construct validity: establishing a chain of evidence to let the study be reviewed and replicated in other settings, and exposing our results to the judgment of people who will be affected by the decisions that our assessment will influence.

We invite other researchers or practitioners who have already proposed or used the measurement of the same or similar attributes to share their experience and further validate their hypotheses with legacy systems in the domain of public administration. We also want to establish links with research groups, government organizations, and industries, working on shared information systems that are interested to replicate our study and tailor our models in their specific domain.

References

[Brod95] M. L. Brodie and M. Stonebraker, *Migrating Legacy Systems - Gateways, interfaces & the incremental approach*, Morgan Kaufmann Publishers, Inc., San Francisco, California, 1995

[Broo95] F. P. Brooks, Jr., *The Mythical Man-Month : Essays on Software Engineering*, anniversary edition, Addison Wesley, 1995.

[Canf97] G. Canfora and A. Cimitile, "A reference life-cycle for legacy systems", *Workshop on Migration Strategies for Legacy Systems,* Boston, May 1997.

[IEEE93] ANSI/IEEE Std 1219, *IEEE Standard for Software Maintenance*, 1993.

[Judd91] C. M. Judd, E. R. Smith, and L. H. Kidder, *Research Methods in Social Relations*, Sixth edition, Harcourt Brace Jovanovich College Publishers, 1991.

[Lehm85] M. M. Lehman, and L. A. Belady, *Program Evolution: Processes of Software Change*, London: Academic Press, 1985.

[Lew97] T. Lewis, "Absorb and extend: resistance is futile!", Computer, vol. 30, no. 5, May 1997, pp.109-112.

[Parn72] D. L. Parnas, "On the criteria to be used in decomposing systems into modules", *Comm. of the ACM*, vol. 15, no. 12, Dec. 1972, pp. 1053-1058.

[Parn94] D. L. Parnas, "Software aging", *Proc. Int. Conf. on Soft. Eng.*, Sorrento, Italy, 1994, pp.279-287.

[Rajl96] V. Rajlich, and J. H. Silva, "Evolution and reuse of orthogonal architecture", *IEEE Trans. on Soft. Eng.*, vol. 22, no. 2, February 1996, pp.153-157.

[Shaw96] M. Shaw, and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline*, Prentice Hall, 1996.

[Snee96] H.M. Sneed "Planning the reengineering of legacy systems", *IEEE Software*, January 1996, pp. 24-34.

[Soni95] D. Soni, R. L. Nord, and C. Hofmeister, "Software architecture in industrial applications", *Proc. Int. Conf. on Soft. Eng.*, Seattle, Washington, 1995, pp.196-207.

[Yin94] R. K. Yin, *Case Study Research: Design and Methods*, second edition, SAGE Publications, 1994.