

Replacement Decisions for *E*-type Software - Some Elements

M M Lehman G Kahen J F Ramil

Department of Computing
Imperial College

180 Queen's Gate, London SW7 2BZ
{mml,gk,ramil}@doc.ic.ac.uk

ICSE'2000 Second Workshop on Economics-Driven Software Engineering Research,
Limerick, Ireland, June 2000

Abstract

E-type software, software implementing real world computer applications, must be continually *evolved* to maintain user satisfaction, generally over several years and releases. It is likely that the responsible organisation will, sooner or later, consider its *replacement* through redevelopment or acquisition. This paper discusses elements believed to be relevant in the context of replacement decisions, involving a phenomenology of software evolution and economic concepts.

Keywords

Software Evolution, Software Replacement, Cost, Economics, Decision Making

Introduction

To maintain the satisfaction of users in general and stakeholder satisfaction in particular, *E*-type software¹, that is software actively used in a real world domain, requires to be continually *evolved*, that is fixed, enhanced and adapted [leh78]. Evidence first identified in the seventies [leh74,8] indicates, *inter alia*, that as a software system is evolved it tends to degrade. This conclusion is supported by observations made during the FEAST [fea00] and other studies [par94] and suggests that evolving software eventually, reaches a condition where, from an economic point of view at least, replacement is indicated. When this occurs, several alternatives arise, for example:

- Continue evolution of the existing system, even though this appears uneconomic
- *Off-the-shelf* replacement
- Acquiring a replacement system whose successful use depends on modifying that system or making changes to the operation/application
- Developing a new system with, possibly, some re-use of old system components.

In what follows, any or several of these alternatives being seriously considered by an organisation will be referred to in singular as the *alternative*. An optimal decision may need to consider *many* factors. Not all related directly to economic considerations. But what they are and how a decision is reached cannot be discussed further here. If *replacement decision* is seen as the way forward one must ask "when is the most appropriate time for replacement, how may one determine it?". But even before replacement is adopted as the selected option the alternatives of slowing down system evolution, temporarily suspending it or replacing parts of the system will have been considered.

Decision making will benefit from availability of *models* that reflect relevant attributes of both the software and many domains within which the system under consideration is being developed and used. These domains include, but are not limited to, the evolution, operation and application domains. The models must also reflect the dynamic properties of these systems and domains and of the interaction between all the entities. Overlooking these may lead to inappropriate decisions. More generally, such decisions cannot be taken on the sole basis of technological and economic factors. As for many others areas, the problems addressed are of socio-technological-economic character [leh80a, kah96]. Thus the ensuing discussion assumes that in resolving the software replacement issue, and as part of the economic assessment, relevant social, technological and business factors are also considered. And in considering them, it must be noted that the latter are also changing and evolving.

¹ The reference to *software* applies here to source code, its documentation and to other documents such as specifications, design, test plans and test cases).

E-type Software Evolution Processes

Replacement decision has been treated before in the context of software maintenance [sak94] and section "Related work" below. To the knowledge of the authors, however, this paper reports on a first, preliminary, attempt to treat the problem in the context of *software evolution*, a wider, encompassing, concept. In this connection, note that maintenance in the sense of fault fixing, enhancement and functional extension are all applied to an already operational system and are included in the term *software evolution* as considered here. That phrase then includes all (on-going) activity that aims to maintain *user satisfaction* over time. The software evolution process is the process by which such activity is accomplished.

The present discussion applies exclusively to *E*-type systems. These are defined as systems embedded and actively used in a real world domain [leh80b]. Assessment of their performance and quality, in terms of such factors as error rate, stability, performance, etc., is based on their behaviour under operational conditions. This basis for acceptability replaces the concept of *correctness* that is the criterion of success in the development of *S*-type systems [leh80b] With few exceptions, business and organisational systems are of type *E* though they may be built from *S*-type elements. Thus, the restriction to the latter does not imply a significant loss in generality of the discussion that follows.

Software Evolution Phenomenology

A 1968 study [leh69] initiated a prolonged investigation of the growth of OS/360-370 and other systems. It led over a period of time to, *inter alia*, the identification of a series of eight behavioural descriptors that were termed *laws of software evolution*. The term *laws* was used because the observed patterns encapsulate common gross behaviour that is, from the perspective of developers, external to the technical development process. It should be noted that identification and refinement of these laws followed the accepted scientific paradigm. A phenomenon is recognised. Metric data to describe it is obtained. Simple models of that data are created. The observed phenomenology is interpreted in terms of the models and *vice versa* and expressed in a formal statement or descriptive text. Understanding of the observed behaviour, the models and the formal statement or descriptive text are then refined and/or extended in an iterative process of successive refinement [leh80a].

The laws were numbered in the order of their formulation. They have been modified over the years as understanding of the phenomena encapsulated in them has advanced [leh74,85,98]. The current text of the laws is presented in Table 1, though definitions of the concepts involved and further explanation cannot be included here.

No.	Brief Name	Law
I	Continuing Change	<i>E</i> -type systems that are in regular use must be continually adapted else they become progressively less satisfactory.
II	Increasing Complexity	As an <i>E</i> -type system is evolved its complexity increases unless work is done to maintain or reduce it.
III	Self Regulation	Global <i>E</i> -type system evolution processes are self-regulating.
IV	Conservation of Organizational Stability	The global activity rate in an evolving <i>E</i> -type system tends to remain constant over product lifetime.
V	Conservation of Familiarity	On average, incremental growth tends to remain constant or to decline.
VI	Continuing Growth	Functional content of <i>E</i> -type systems must be continually increased to maintain user satisfaction over the system lifetime.
VII	Declining Quality	The quality of an <i>E</i> -type system will decline unless it is adapted to take into account changes in its operational environment.
VIII	Feedback System	<i>E</i> -type evolution processes are multi level, multi loop, multi agent feedback systems and must, in general, be treated as such to achieve major process improvement for other than the most primitive processes.

Table 1 - The Laws of Software Evolution

A series of four models [bel75,rio77,woo79,ram00] relate to the problem of software evolution costs and lifetime estimation. These models identified growing software complexity as a major obstacle to achieving successful and continuing software evolution. All four models involved the concept of *anti-regressive* work [leh74]. So was termed the effort, such as complexity control, that did not increase system performance or functionality, but did, for example, improve its maintainability. The models indicate that the economic lifetime of *E*-type software that is in use is limited, at least in part, by *growing complexity* and the *anti-regressive policy* that is followed in seeking to maintain or reduce it. In the simulation models mentioned above [bel75,rio77,woo79,ram00] neglect of the anti-regressive work reduces the economically, useful, lifetime. Different policies in this regard yield different software lifetimes and, indeed, different process productivity profiles. Thus, one has to consider this aspect when seeking to understand long term software evolution costs.

Other factors playing a role in the determination of software lifetime are *workload* and *work pressure*. The almost inevitable consequence of excessive work pressure is, for example, structural deterioration, coding errors and decay in the precision and completeness of the documentation. The evidence suggests that it should be beneficial to follow periods of intensive evolution by clean up periods during which anti-regressive such as restructuring is undertaken. Enhancement and functional change should be suppressed or, at worst, minimised during such periods. Such a strategy has, indeed been adopted in the past by various organisations. Woodside model also provides support to such strategy [woo79].

From this brief discussion it follows that the economic lifetime of software depends, *inter alia*, on the adopted software evolution policy. If complexity and other software aging effects, together with workload management and software process issues are appropriately managed a longer software lifetime is likely to be achieved. If maximisation of user satisfaction is an objective in the evolution of a software system, for maximum effectiveness the planners and replacement decision makers must consider the relationship between measures of complexity, their rate of change, cost/benefit and other factors.

Another element must be also mentioned. The evolution process reaches well beyond the development organisation. The process is a *global process* that includes all agents whose role affects user satisfaction. Such agents include developers, users, support personnel, marketeers and managers of the many organisations that influence or participate in the process, in fixing its goals and the approval and transfer of its products [fea00]. The consequent interactions across organisational boundaries and their interacting evolutionary trends, impose a serious challenges to quantification and evaluation of costs and benefits to help achieve well founded and informed replacement decisions. It follows that any discussion that refers to the cost and benefit of software evolution requires the establishment of an arbitrary process boundary by the analyst. Determination of the appropriate bounds may vary widely from case to case. As a starting point one may focus on the costs of the *technical* software evolution process only, that is, the process that services a flow of work requests, and generates a continuing flow of evolved, that is fixed, enhanced, adapted or new functionality. That is necessary but not, in the long run, sufficient.

Some Economic Elements

What follows is a brief description of the concepts we have been considering in the examination of the *E*-type software replacement time. These concepts may need refinement and others may be included as the investigation progresses. In this analysis, time is considered in discrete units or intervals.

Sunk costs - software replacement decision can be approached by excluding from the decision making any consideration of sunk costs [bes00]. The latter are generally defined as unavoidable costs or cost already incurred and that, by being so, are not affected by the decision outcome. As an example consider the cost of evolving a system to the present time or to the current release. This would, normally, not be considered in a decision concerning the future of that system.

Cash flow for current(old) system $PV_o(i,j)$ - *Present value*² at time interval *i* of sequence of benefits minus costs associated to the current system supporting the application over the evaluation period from interval *i* to *j*, inclusive.

² The *present value* PV penalises future income or expenses by using a discount rate *k*. So, for a sequence of future periodically incurred costs or received benefits, PV is given by $\sum_{i=1}^n F_i/(1+k)^i$ where F_i is the cash flow of interval(e.g. year) *j*, *k* and *N* is the total number of intervals being considered.

Cash flow for alternative(new) system $PV_n(i,j)$ - Present value at time interval i of sequence of benefits minus costs associated to the alternative system supporting the application over the evaluation period from interval i to j , inclusive.

Time horizon N_h - Maximum number of time intervals considered in the evaluation of the present value. Generally, N_h will represent the remaining application lifetime, the period over which the application will have to be supported by a software system, either the current or the alternative. In principle, it offers a reasonable time horizon for the analysis of the replacement decision. A simplifying assumption is that N_h is the same for the current and for the alternative system. This is a conservative approach since one may reasonably expect that the replacement alternative, being based on increased application experience and more recent technology may be practically superior. Such improvements may result in an increase of the application life expectancy and/or a more powerful application later on, in either case resulting in increased benefit associated to the alternative.

Development intervals for alternative N_i - Number of time intervals from start of the development of the alternative to effective replacement of the existing software by the alternative.

Replacement interval N_r - Earliest time at which the *alternative* represents the best choice, according for example, to the criterion $PV_o(N_r, N_h) < PV_n(N_r, N_h)$ being satisfied.

Algorithm for replacement decision -

1. Start at i = present or future time interval desired as starting point of analysis³
2. Estimate or re-estimate time horizon N_h , and discount factor k .
3. Estimate cash flow associated to *current* system $PV_o(i, N_h)$
4. Estimate cash flow associated to *alternative* system $PV_n(i, N_h)$
5. If $PV_o(i, N_h) < PV_n(i, N_h)$ then (accept $N_r=i$ as appropriate time for starting development of *alternative* system; proceed to replace system with alternative at interval $i+N_r$), otherwise: set $i = i + 1$ and go to step 2.

Further Work

A system dynamic model such as the one recently developed by these authors [ram00] suggests an approach estimate of costs associated to the current and the alternative systems. One would need to adapt, refine and calibrate one instance of the model to the *current* system and another to the *alternative*. To translate the model in economic terms, one must, however, incorporate *present value* calculations into the present dynamic models [ram00]. The work by Coyle and Rego [coy82] in a different domain appears to offer solution to some of the modelling challenges we foresee. This, however, has not been accomplished. Estimation of benefits appears as a major challenge unless historical data is available. Modelling techniques developed for marketing [lil92] could be used to address this aspect. Validation of simulation and of replacement decision results may be addressed by comparison with that reached by following approaches suggested by others [sat94,cha96].

The decision approaches mentioned above [sat94,cha96] and the proposed in this paper are deterministic in nature. Uncertainty is intrinsic to the costs, benefits and intervals involved in the replacement decision. This may be addressed by incorporating Monte Carlo simulations in the models used to derive costs and benefits. Possible ways to refine the proposed algorithm is by incorporation of elements from stochastic hypotheses testing [bas93].

Acknowledgements

The authors gratefully acknowledge financial support from the UK EPSRC, FEAST/2 project, grant number GR/M44101.

³ This is to avoid consideration of past costs in the analysis, which are sunk costs.

References⁴

- [bes00] Besanko D, Dravone D and Shanley M, *Economics of Strategy*, Wiley, NY, 2000
- [boe81] Boehm B, *Software Engineering Economics*, Englewood Cliffs, N.J, Prentice-Hall, 1981
- [bas93] Basseville M and Nikiforov IV, *Detection of Abrupt Changes: Theory and Application*, Prentice Hall, Englewood Cliffs, NJ, 1993, 528 p
- [bel75] Belady LA and Lehman MM, *The Evolution Dynamics of Large Programs*, IBM Systems Journal, Vol. 15, No. 3, 1976
- [cha96] Chan T, Chung SL and Ho TH, *An Economic Model to Estimate Software Rewriting and Replacement Times*, IEEE Trans. on Softw. Engineering, Vol. 22, No. 8, August 1996, pp. 580 - 598
- [coy82] Coyle RG and Rego JC, *Modelling the Discrete Ordering of Hydro-electric Projects - The Argentinian Case*, Dynamica, Vol. 8, 1, Summer, 1982
- [coy96] Coyle RG, *System Dynamics Modelling - A Practical Approach*, Chapman & Hall, London, 1996, 413 p
- [feh00] Feedback, Evolution and Software Technology, FEAST/2 Project Web Site, <http://www-dse.doc.ic.ac.uk/~mml/feast/>
- [kah96] Kahen G, *Building a Framework for Successful Information Technology Transfer*, International Journal of Computer Applications in Technologies, Vol. 6, No. 1, pp. 1 - 8
- [leh69]* Lehman MM, *The Programming Process*, IBM Res. Rep. RC 2722, IBM Res. Centre, Yorktown Heights, NY 10594, Sept. 1969.
- [leh74]* Lehman MM, *Programs, Cities, Students, Limits to Growth?*, Inaugural Lecture, May 1974. Publ. in Imp. Col of Sc. Tech. Inaug. Lect. Ser., Vol. 9, 1970, 1974, pp. 211-229. Also in Programming Methodology, (D Gries ed.), Springer, Verlag, 1978, pp. 42-62.
- [leh78]* id, *Laws of Program Evolution - Rules and Tools of Programming Management*, Proc. Infotech State of the Art Conf., Why Software Projects Fail?, Apr. 1978, pp 11/1 - 11/25
- [leh80a]* id, *On Understanding Laws, Evolution and Conservation in the Large Program Life-Cycle*, Journal of Systems and Software, Vol. 1, No. 3, 1980
- [leh80b]* id, *Programs, Life Cycles and Laws of Software Evolution*, Proc. IEEE Spec Iss on Softw. Eng., Sept. 1980, pp 1060 - 1076, With more detail as *Programs, Programming and the Software Life-Cycle* in System Design Infotech State of the Art Report, Se 6, No 9, Pergamon Infotech Ltd, Maidenhead, 1981, pp 263 291
- [leh85] Lehman MM and Belady LA, *Program Evolution - Processes of Software Change*, Academic Press, 1985, pp 538
- [leh98] Lehman M.M., D.E. Perry, and J.F. Ramil, *On Evidence Supporting the FEAST Hypothesis and the Laws of Software Evolution*, Proc. of Fifth Int. Metrics Symposium, Metrics'98, Bethesda, Maryland, Nov. 20-21, 1998.
- [lil92] Lilien GL, Kotler P and Moorthy KS, *Marketing Models*, Prentice Hall, Englewood Cliffs, NJ, 1992
- [par94] Parnas DL, *Software Aging*, Proc. 16th Int. Conf. on Softw. Engineering, May 16-21, 1994, Sorrento, Italy, pp 279-287
- [rio77] Riordan JS, *An Evolution Dynamics Model of Software Systems Development*, in Software Phenomenology - Working Papers of the (First) SLCM Workshop, Airlie, Virginia, Aug 1977. Pub. ISRAD/AIRMICS, Comp Sys Comm US Army, Fort Belvoir, VI, Dec 1977 , pp 339 - 360
- [ram00] Ramil JF, Lehman MM and Kahen G, *The FEAST Approach to Quantitative Process Modelling of Software Evolution Processes*, submitted for publication, Feb. 2000. Also as Kahen G, Lehman MM and Ramil JF, Model-based Assessment of Software Evolution Processes, Research Report, Dept. of Computing, Imperial College, February 2000, 20 pp
- [sak94] Sakthivel S, *A Decision Model to Choose between Software Maintenance and Software Redevelopment*, Software Maintenance: Research and Practice, Vol. 6, 1994, pp. 121 - 143
- [woo79]* Woodside CM, *A Mathematical Model for the Evolution of Software*, ICST CCD Res. Rep 79/55, Apr. 1979. Also in J Sys and Software, Vol 1, No 4, Oct 1980, pp 337 - 345, and as Chapter 16 in Lehman MM and Belady LA, Program Evolution - Processes of Software Change, Academic Press, 1985, pp 538

⁴ References with an "*" have been reprinted in [leh85].