Replacement Decisions for Evolving Software

M M Lehman J F Ramil G Kahen
Department of Computing
Imperial College
180 Queen's Gate, London SW7 2BZ
tel +44 20 7594 8214
fax +44 20 7594 8215
{mml,ramil,gk}@doc.ic.ac.uk

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. A procedure to obtain the optimal replacement time based on *present value* calculations is suggested.

Keywords
E-type software, software evolution, software replacement decisions, cost estimation, economics

1 INTRODUCTION

To maintain the satisfaction of users in general and stakeholder satisfaction in particular, *E*-type software[1], that is software actively used in a real world domain, requires to be continually *evolved*, that is fixed, enhanced and adapted [11,14]. Evidence first identified in the seventies [10,11] indicates, *inter alia*, that as a software system is evolved it tends in several aspects to degrade. This conclusion is supported by observations made during the FEAST [7] and other studies [17] 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, that is acquiring a replacement system that initially suits the application without having to modify that system
- 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.

[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.

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 are related directly to economic considerations. But what they are and how a decision is reached cannot be discussed further here. If the *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 [8,12]. 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.

2 *E*-TYPE SOFTWARE PROCESSES

Replacement decision has been treated before in the context of software maintenance, for example in [4,20]. To the knowledge of the authors, however, this paper reports on a first, preliminary, attempt to treat the problem in the context of *evolving* software. Evolution is a wider, encompassing, concept than maintenance. 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 intended to maintain *stakeholders' satisfaction* over time. The software evolution process is the process by which such activity is accomplished. Stakeholders include users, developers, support personnel, marketeers and so on.

The present discussion applies exclusively to *E*-type systems. These are defined as systems embedded and actively used in a real world domain [13,14]. They include, as intrinsic property, the need to be continually evolved. 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 [13,14]. With few exceptions, business and organisational systems are of type *E* though they may be built from *S*-type elements [7]. Thus, the restriction to the latter does not imply a significant loss in generality of what follows.

3 EVOLUTION PHENOMENOLOGY

A 1968 study [9] 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 [12].

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 [7,10,14,15]. The current text of the laws is presented in the appendix (Table 1), though definitions of the concepts involved and further explanation cannot be included here.

A series of four models [3,18,19,21] 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 [10]. 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 [3,18,19,21] 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 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 [21].

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 and, generally, individual managers' decisions. The process is a *global process* that includes all agents whose role affects user satisfaction. Such agents include developers, users, support and sales personnel, 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 [7]. 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.

4 A PROCEDURE TO ACHIEVE AN OPTIMAL REPLACEMENT DECISION

What follows is a brief description of the concepts and a procedure we suggest for the achievement of an optimal decision for the *E*-type software replacement time. These concepts and procedure may need refinement and other concepts may have to be included as the investigation progresses. In this analysis, time is considered in discrete units or intervals, such as years. The concepts are the following:

Sunk costs - software replacement decision can be approached by excluding from the decision making any consideration of sunk costs [1]. 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.

Time horizon N_h - Maximum number of time intervals considered in the evaluation of the cash flows (see below). 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 (or acquisition) 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. In case of an *off-the-shelf* alternative, N_i represents the time needed to perform the acquisition .

Cash flow for current (old) system $PV_o(i,N_h)$ - *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 N_h, inclusive.

Cash flow for alternative (new) system $PV_n(i,N_h)$ - 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 N_h, inclusive. $PV_n(i,N_h)$ must include the estimated development (or acquisition) costs over the appropriate development (or acquisition) subinterval $[i,i+1,...,N_i]$, with N_h always greater than N_i.

[2] The *present value PV* penalises a sequence of future income or expenses by using a discount rate k. So, for a sequence of future incurred costs and/or received benefits from time interval i to n, $PV(i,n)$ is given by $S^{n}_{j=i}\, F_j/(1+k)^j$ where F_j is the cash flow (benefits minus cost) during interval j.

Replacement interval N_r - Earliest time at which the *alternative* represents the best choice, according for example, to the inequality

$$PV_o(N_r,N_h) < PV_n(N_r,N_h) \qquad \text{(Eq. 1)}$$

becoming true, as exemplified by the following procedure. Steps to obtain optimal replacement interval N_r: for a given time horizon N_h, and discount factor k.

1. Start at i equal to time interval desired as starting point of analysis[3]
2. Estimate $PV_o(i,N_h)$, the cash flow from interval i to interval N_h associated to the current system
3. Estimate $PV_n(i,N_h)$, the cash flow from interval i to interval N_h associated to the alternative system
4. 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 $i+N_i$); otherwise: set $i = i + 1$ and go to step 1.

5 FURTHER WORK

A *system dynamic* [6] model such as the one recently developed by the present authors [19] can be used as part of an approach to estimate the cost of further evolution associated to the current and the alternative systems. One would need, for example, 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 [19]. The work by Coyle and Rego [5] 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 appropriate historical data is available. Modelling techniques developed for marketing [16] 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 [4,20].

The decision approaches mentioned above [4,20] and the proposed in this paper are deterministic. However, uncertainty is intrinsic to the costs, benefits and intervals involved in the replacement decision. This may be addressed by incorporating systematic decision procedures that combine qualitative [8] and quantitative elements. Uncertainty can also be addressed by Monte Carlo simulation modelling to derive costs and benefits. For example, a possible way to refine the proposed procedure is by evaluating Eq. 1 as a stochastic decision (hypothesis testing) problem with methods as in [2].

[3] i equal to 1 represents the time interval about to commence. i must be equal or greater than 1, to avoid consideration in the analysis of costs incurred in the past, which are considered *sunk costs*.

ACKNOWLEDGEMENTS
Authors acknowledge financial support from the UK EPSRC, FEAST/2 project, grant number GR/M44101.

REFERENCES[4]

1. Besanko D, Dravone D and Shanley M, *Economics of Strategy*, Wiley, NY, 2000
2. Basseville M and Nikiforov IV, *Detection of Abrupt Changes: Theory and Application*, Prentice Hall, Englewood Cliffs, NJ, 1993, 528 p
3. *Belady LA and Lehman MM, *The Evolution Dynamics of Large Programs*, IBM Systems Journal, Vol. 15, No. 3, 1976, pp. 225 - 252
4. 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, Aug. 1996, pp. 580 - 598
5. Coyle RG and Rego JC, Modelling the Discrete Ordering of Hydro-electric Projects - The Argentinian Case, *Dynamica*, Vol. 8, 1, Summer, 1982
6. Coyle RG, *System Dynamics Modelling - A Practical Approach*, Chapman & Hall, London, 1996, 413 p
7. *Feedback, Evolution and Software Technology*, FEAST/2, <http://www-dse.doc.ic.ac.uk/~mml/feast/>
8. Kahen G, Building a Framework for Successful Information Technology Transfer, *International J. of Comp. Applic. in Technologies*, Vol. 6, No. 1, pp. 1 - 8
9. *Lehman MM, *The Programming Process*, IBM Res. Rep. RC 2722, IBM Res. Centre, Yorktown Heights, NY 10594, Sept. 1969.
10. *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.
11. **id*, Laws of Program Evolution - Rules and Tools for Programming Management, *Proc. Infotech State of the Art Conf.*, Why Software Projects Fail?, Apr. 1978, pp 11/1 - 11/25
12. **id*, On Understanding Laws, Evolution and Conservation in the Large Program Life-Cycle, *Journal of Systems and Software*, Vol. 1, No. 3, 1980
13. **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, Maidenhead, 1981, pp 263 291
14. Lehman MM and Belady LA, *Program Evolution - Processes of Software Change*, Acad. Press, London, 1985, pp 538
15. Lehman MM, Perry DE, and Ramil JF, On Evidence Supporting the FEAST Hypothesis and the Laws of Software Evolution, *Proc. Metrics'98*, Bethesda, Maryland, Nov. 20-21, 1998.
16. Lilien GL, Kotler P and Moorthy KS, *Marketing Models*, Prentice Hall, Englewood Cliffs, NJ, 1992
17. Parnas DL, Software Aging, *Proc. ICSE 16*, May 16-21, 1994, Sorrento, Italy, pp 279-287
18. Ramil JF, Lehman MM and Kahen G, The FEAST Approach to Quantitative Process Modelling of Software Evolution Processes, submitted for publication, to appear in *Proc. PROFES'2000 2nd International Conference on Product Focused Software Process Improvement*, Oulu, Finland, June 20 - 22, 2000, LNCS Springer.
19. 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
20. Sakthivel S, A Decision Model to Choose between Software Maintenance and Software Redevelopment, *Journal of Software Maintenance: Research and Practice*, Vol. 6, 1994, pp. 121 - 143
21. *Woodside CM, A Mathematical Model for the Evolution of Software, ICST CCD Res. Rep 79/55, Apr. 1979. Also in *J.l of Sys. and Softw.*, Vol 1, No 4, Oct 1980, pp 337 - 345

[4] References with "*" have been reprinted in [14].

APPENDIX

No.	*Brief Name*	*Law*
I	Continuing Change	*E*-type systems that are in regular use must be continually adapted else they become progressively less satisfactory.
II	Increasing Complexity	As an *E*-type system is evolved its complexity increases unless work is done to maintain or reduce it.
III	Self Regulation	Global *E*-type system evolution processes are self-regulating.
IV	Conservation of Organizational Stability	The activity rate of *E*-type evolution processes, in appropriate measures, tends to remain constant over long periods of product evolution.
V	Conservation of Familiarity	Average incremental growth of *E*-type systems over a sequence of releases or equal time intervals tends to decline.
VI	Continuing Growth	The functional content of *E*-type systems must be continually increased to maintain user satisfaction over the system lifetime.
VII	Declining Quality	Unless rigorously adapted to take into account changes in the operational environment, the quality of *E*-type systems will appear to be declining
VIII	Feedback System	*E*-type evolution processes are multi level, multi loop, multi agent feedback systems

Table 1 - Current Statement of the Laws of Software Evolution