

Feedback, Evolution And Software Technology - Some Results from the FEAST/1 Project

Keynote Address

11th International Conference Software Engineering & its Applications
Paris, December 8, 1998

M M Lehman J F Ramil

Department of Computing
Imperial College
London SW7 2BZ

tel: +44 (0)171 594 8214 fax: +44 (0)171 594 8215

{mml,jcf1}@doc.ic.ac.uk
<http://www-dse.doc.ic.ac.uk/~mml>

Abstract

Modelling and analysis of the evolution of several industrial software systems, on the basis of empirical data obtained as part of the recently completed FEAST/1 research project, confirm conclusions from an early seventies study of IBM OS/360-370. In particular, they are largely consistent with the laws of software evolution developed from the results of the earlier study. FEAST/1 results also support the view that *global* development and evolution processes of real world software (*E*-type) are multi-agent, multi-loop, multi-level feedback systems and reinforce the assertion that the feedback nature of the software process has to be taken into account in mastering that process and in achieving its sustainable improvement. This paper concentrates on growth data and provides examples of theoretical and empirical evidence of feedback based on a subset of the results obtained, indicating also issues requiring further elucidation.

1 Introduction

The software development and maintenance process, more appropriately described jointly as the software *evolution* process [leh84], constitutes a feedback system. This observation was first made in a 1972 paper [bel72] discussing some results of a study of the IBM programming process [leh69]. It observed that the ripple in the plot of OS/360-370 growth shown in figure 1, "... is typical of a self stabilising process with positive and negative *feedback* loops. ... the rate of system growth is self-regulatory, despite the fact that many different causes control the selection of work implemented in each release, with varying budgets, increasing numbers of users reporting faults or desiring new capability, varying management attitudes towards system enhancement, changing release intervals and improving methods ...".

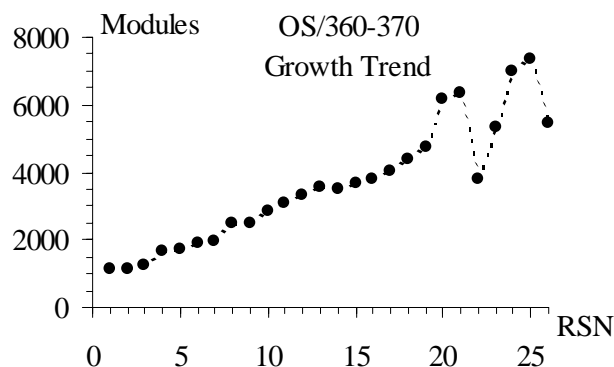


Figure 1 IBM OS 360/370 Growth Trend

Subsequently it was suggested [leh78] that one must "... regard the organisation developing and maintaining a large program as a system in the system theoretic sense. ... Observation has shown that the system behaves as a self stabilising feedback system. The process leads to an organisation and a process dominated by feedback ... with long range trends ... and invariances ...". It was these

observations and the analysis and interpretation that followed that led to a deeper understanding of the nature and phenomenology of software evolution as reflected in three, then five and, ultimately, eight *Laws of Software Evolution* [leh74,80,96a] and a *Principle of Uncertainty* [leh89]. And until 1994 that was how the matter was left.

More recently one of the authors, while considering the *total* software process¹ asked "Why is it so difficult to achieve major tangible progress in process improvement?" Once asked, a possible answer immediately came to mind. Since its beginnings, improvement of the software process has concentrated primarily on the *forward* path of the feedback system constituted by the process. This is exemplified by the introduction of new steps (eg. requirements definition), methods (formal or otherwise) and support tools, for example, into the process. All these are process local and have produced local improvement. Their impact on the wider process is less apparent, may be counter intuitive and is indeed often questioned, eg. [abd91, you98].

In term of the feedback hypothesis the explanation is self-evident. Being contained within a number of *feedback* loops, some of them negative, the impact of forward path improvements at the global level will generally be constrained. Note, in this context, that the *global* software process includes the activities of technical development, but also those of local and corporate management, marketing, sales, user support, users and many others [leh96d]. To change the behaviour of a feedback system *as viewed from the outside* requires it to be treated as such. In the absence of changes to the feedback mechanisms the impact of forward path changes on total system behaviour is likely to be attenuated. The nature and degree of attenuation cannot be predicted without detailed knowledge of, for example, the topology of the feedback network and the information modification, gain and delays in the various loops. The realisation that the process constitutes a multi level, multi loop, multi agent feedback system implies that it is essential to take the role of feedback into account to obtain a degree of intellectual control of the software process, to master it and to achieve its sustained improvement. Most recently it has been realised that given such a structure, constraints arising from mechanisms in the outer loops are likely to be dominant. This assertion and evidence to support it is briefly discussed in section 5.

Note that all references in this paper to software systems are limited to those of type *E*, that is, those which are to be used in the real-world [leh85]. Similarly, references in this paper to the software process refer to the *global* software process, as described in the preceding paragraph. In the remainder of this paper evidence and phenomenology in support of the assertion that *E*-type software processes are feedback systems is presented. The conclusions are based both on theoretical reasoning and on results obtained from the recently completed FEAST/1 project.

2 Feedback in the Global Software Process

As stated above, software processes are intrinsically feedback systems. Feedback mechanisms and influences appear at many levels and in a variety of forms. Previous studies eg. [abd91, mad96] have concentrated mainly on developing system dynamics models capturing the behaviour of specific activities within some specified context. The discussion of this section will concentrate on characteristics which are believed to be common to *E*-type software processes, in general. The discussion applies both to *ab initio* development of a software system and to the evolution of such a system to achieve a new version, enhanced in some way.

E-type software operates in an unbounded countably infinite real world domain [leh89]. Its development² is initiated by explicit or implicit formulation of an *application concept* or a *change* to an existing concept. The formulation includes the setting of bounds both to the concept and to its operational domain. Such bounds are inevitable because the software implementing an application is necessarily bounded and finite. Thus the software system as a model of the application in its

¹ The *total process* is the sum total of all activity required to transform an initial computer application concept or a change to an existing concept or its implementation to installation and operation in the intended operational domain.

² The term *development* is used hereinafter to include both *ab initio* development and change to an existing system.

operational domain is essentially incomplete. The resultant gap between the finite operational software and the infinite real world domain is bridged by assumptions reflected, for example, in specifications, design details, algorithms, implementations, parameter values and test conditions.

Application bounding involves many stake holders, clients, users, managers, marketeers, development staff and so on. Their different views must be elicited, reconciled, merged and limited to provide a system definition, whose implementation is technically feasible, within time, resource, budget and other constraints. A process of successive blending and reconciliation of the many viewpoints provides a starting point for the development and subsequent evolution of the software system. This essentially iterative process is illustrated in figure 2. The process moves the bounds of both the application and its domain until they converge to a view that is acceptable to the decision makers. The latter base their judgement on their understanding of the expectations of the various stake holders. This involves assessment of the degree to which satisfaction of the individual expectations supports the purpose of the development, however defined.

The bounding process has the structure of a classical feedback loop in which the output of a process is applied to and modifies its input. Once the decision to go ahead has been made further development steps can be pursued to produce a validated system ready for installation. Figure 3 exemplifies the sequence of actions required, identifying basic steps required to develop a satisfactory operational system. The actual set of steps and overall process followed will vary from organisation to organisation and the figure is not intended to establish a definitive process model.

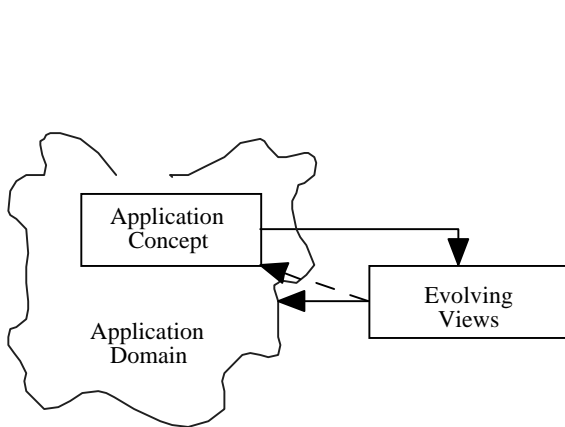


Figure 2 Minor Loop

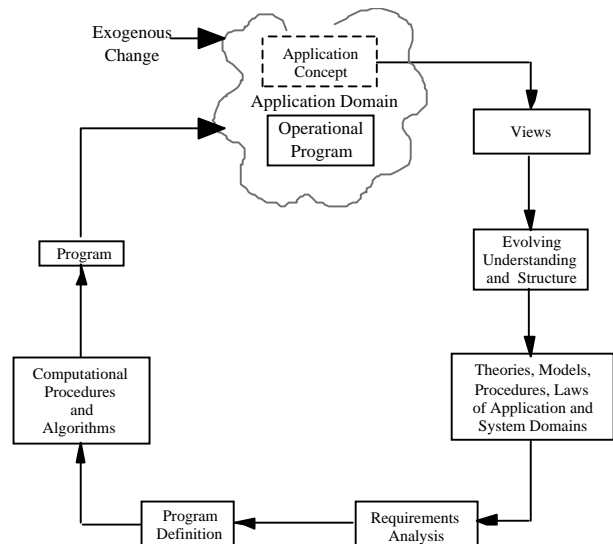


Figure 3 Intermediate Loop

The end product of this or an equivalent process is the validated program. It is installed in the application domain as shown in figure 3. Such installation *changes* the operational *domain*. In addition, installation and use of the software system changes the perceptions, understanding and goals of both users and developers. As the system is operated new opportunities are recognised, activities and roles within the application domain change, new ones emerge, experience is gained and technology advances [leh85]. Moreover, the installed software becomes a *de facto* part of the operational domain. Conceptually at least, the software includes an (implicit) model of itself [leh89]. This too implies the essential imprecision of the software system as a model of the application concept in its operational domain. More practically, it implies continual pressure for system adaptation and extension. In summary, the application and its operational domain that provided the initial input to the development process is changed by the output of that process. At the level illustrated in figure 3 too the software development constitutes a feedback system.

As already mentioned, the gap between the finite software and the unbounded domains it models is bridged by assumptions. Moreover, the real world operational domain is always changing at a rate that, in most of the cases, is accelerated by the installation of the operational software. This implies that individual members of the assumption set progressively become invalid. This results in an unending stream of corrective or adaptive changes. This constitutes a source of positive feedback in that it produces pressure for change, growth in the queue of change requests. The resultant pressure for system growth is management moderated, by checks and balances for example. A part, at least, of these controls represent negative feedback mechanisms.

This brief phenomenological analysis rationalises the universal experience since the beginning of the computer age for continual software maintenance, unending software evolution. In passing it may be noted that the introduction of formal specifications cannot overcome the behaviour outlined above. Clearly, they can capture the expressed bounded properties of an application and its intended operational domain *at a given moment in time*. They may even express recognised or expected directions of change. They cannot address the *actual*, the unforeseen and often unforeseeable changes as they occur in the future. Thus their effectiveness in managing evolution is limited.

The process of invalidation of real world assumptions inherent in all *E*-type development has one further implication. However often a system has run and produced satisfactory results, its next run may fail, for example, as a consequence of an assumption that has become invalid or of encountering, for the first time, a condition that violates an embedded assumption. As expressed in a *Principle of Software Uncertainty* [leh89], the outcome of any *E*-type software execution and its effect in the operational domain is not absolutely predictable. It is intrinsically uncertain.

Figures 2 and 3 provide examples of feedback at the technical step and the release cycle level of the software evolution process. Figure 4 depicts a still higher level of feedback, one that illustrates sources of various driving forces, management checks and balances, controls and other influences on and in the process. They are exercised, consciously or unconsciously, by local and executive management, by marketeers, support personnel, users and many other people and processes in the operational and development domains. All involve feedback loops, in one way or another.

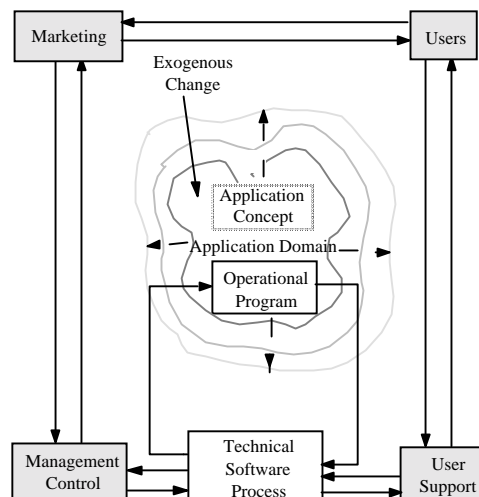


Figure 4 Major Loops

The inner cloud is intended to symbolise the bound of the operational domain at some moment in time. But as indicated above, and as implied by the first and sixth laws of software evolution (Table 1), a fundamental property of *E*-type applications is that this boundary changes as the system is adapted to remove functional and performance limitations and to support a changing environment. The dotted arrow and shadowy clouds are intended to convey this fundamental property of *E*-type systems, that *E*-type systems constitute expanding universes.

Finally, the three levels represented by figures 2, 3 and 4 are merged in figure 5 to provide a still wider and more picturesque view of the global process. The figure seeks to convey the learning, communication and iteration that is an integral part of each of the steps and between the steps. Figure 5 is an attempt to portray the global software process as a complex multi-agent, multi-loop, multi-level feedback system. It does not, however reflect the unceasing evolution which could, for example, be portrayed by a sequence of such pictures one behind the other enveloped in clouds of ever increasing size.

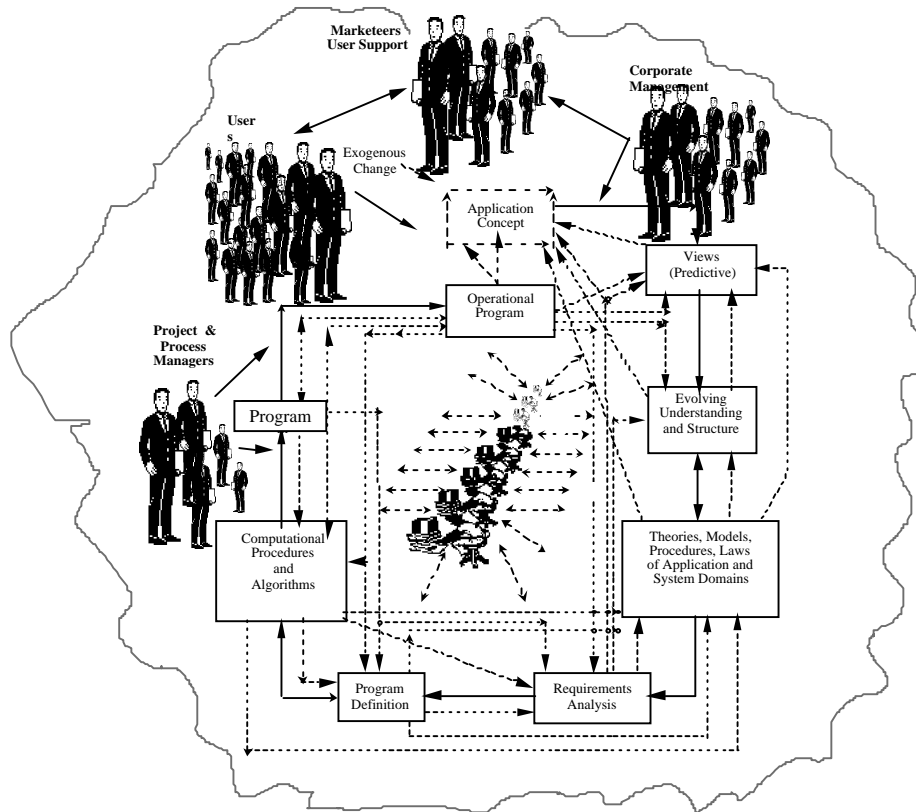


Figure 5 Feedback in the Global Software Process

3 Feedback and the Laws of Software Evolution

An earlier version of the growth trend shown in figure 1 led to the observation that software evolution processes are feedback systems. It was first identified in a study of the IBM programming process [leh69]. That study initiated a prolonged investigation of the growth of OS/360-370 and other systems, leading over a period of time to a series of eight laws of software evolution.

These statements were termed *laws* since they encapsulate behaviour and phenomena which, from the perspective of developers, are external to the technical development process. It should be noted that identification and refinement of these laws has 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 refined and/or extended in an iterative process of successive refinement.

The laws were numbered in the order of their formulation and have been modified over the years as understanding of the phenomena encapsulated in them has advanced. The eighth law provides a formal statement of the feedback system nature of software *development* and *maintenance* processes. It is now apparent that the other seven encapsulate various aspects of interplaying feedback influences. Formulation of the eighth as a law was based on the reasoning outlined in the previous section, results from empirical studies and many years of observations of the programming

process. By treating the law as a research hypothesis, the recently completed FEAST/1 project, has, through its results, increased confidence in its inclusion as a law.

No.	Brief Name	Law
I	Continuing Change	<i>E</i> -type systems 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 Organisational Stability	The average effective global activity rate in an evolving <i>E</i> -type system tends to remain constant over the product lifetime.
V	Conservation of Familiarity	On average, incremental growth tends to remain constant or to decline.
VI	Continuing Growth	The functional content of <i>E</i> -type systems must be continually increased to maintain user satisfaction over their lifetime.
VII	Declining Quality	The quality of <i>E</i> -type systems will appear to be declining unless they are rigorously maintained and adapted to operational environment changes.
VIII	Feedback System	<i>E</i> -type evolution processes constitute multi-level, multi-loop, multi-agent feedback systems and must, in general, be treated as such to achieve significant process improvement for other than the most primitive processes.

Table 1 Laws of Software Evolution

4 FEAST/1

The eighth law was discussed as a hypothesis in a series of three international FEAST (*Feedback, Evolution And Software Technology*) workshops [leh94,95], leading in 1996 to the FEAST/1 project. The proposal for that project [leh96c] recognised that practical exploitation of the eighth law, as expressed in the hypothesis, and its implications required more than the evidence from OS/360-370 and other systems studied during the seventies. The adopted project goals were, therefore, to search for further empirical evidence in several industrial software evolution projects, to identify specific feedback loops and mechanisms and to model and measure their impact on the short and long term evolutionary trends of the respective systems. The industrial organisations involved and the systems/projects they offered for study included British Aerospace and a real time defence system, ICL and the Kernel of the VME operating system, Logica and the FW banking transaction system, Lucent and two variants of a very large real time system and the UK MoD.

The study was conducted by retrieving historical data on the evolution of each of the systems, constructing, analysing and interpreting black box models of each, and reconciliation of the observations with the observed phenomenology and with earlier results such as the other seven laws of software evolution.

A further activity involved the development, calibration and validation of white box system dynamics models [ven95] of two of the processes/systems studied (BAe and ICL). These models made possible the simulation of high level evolutionary trends of these systems [leh98a, wer98]. In each case, the parent organisations consider the resultant models as providing valid, and to them partially new, perspectives on their processes, providing indications of how their processes might be improved.

In brief, the set of results from the black and white box studies have revealed, for seven of the eight laws, behaviour that is, after minor adjustments of their formulation, consistent with the influence and impact to be expected from feedback mechanisms. The seventh law is neither supported nor negated. Results are contained in a number of publications [leh97a,b,98b,d,e,f,fea98]. The next section summarises one set of results illustrating them with data from some of the systems studied. These may be taken as generally typical of the results obtained from the other analyses.

5 An Example of the Black Box Results

There are many indicators and potentially useful metrics of software evolution. From the point of view of the users, for example, measures of functionality, system capability and system power are relevant. Other attributes which one would like to quantify include complexity, reliability, ease of use, quality and cost. From the point of view of developers, complexity and changeability (ease of evolution) are of particular interest. Each of the metrics can provide insight into the evolution process. Taken together they provide supportive and complementary views. Useful metrics involve, however, many trade offs and are not easy to define or measure [leh98d]. In any event, for studies based on archives, such as those associated with configuration control systems, one has to make do with whatever data is available. Moreover, one must recognise that definitions of data collected under a common label may well have changed over the years. Despite such difficulties, and as illustrated by the results that follow, meaningful results may be obtained.

One measure universally available in historic records is the size of successive versions of the system as recorded at release. This may have been counted as numbers of modules, files, classes, function points, holons and so on. At another (though in practice less useful) level the size of the system may be expressed in lines of code (locs). The definition of these terms will vary from organisation and from time to time. Where only one such metric is available for a given system the analyst has no option but to use it. Where several are available the same analysis can be performed on each and the results compared. Such comparisons as have been undertaken suggest that, at least at the level of detail reported here, similar system behaviour is generally observed. In the report that follows all size measurement will relate to numbers of modules (however defined in each organisation), a measure that has produced consistent results over the years.

Figure 6 shows the growth of the FW banking transaction system and of one of two variants of a Lucent Technology large real-time system. The growth is expressed relative to the size of the first data point, 977 modules for the former and 44,471 modules for the latter. The FW data corresponds to over 5 years of system evolution, the Lucent data to over 15. The growth is displayed over software releases, each identified by an integer termed *release sequence number* (RSN). In between releases, the system is generally in state of flux, undergoing continuous change. Moreover, real world software evolution often involves parallel development lines. The decision as to what is to be included in a release may only be taken shortly before release time and work under way at any moment will be producing modules for several releases. In general only at release time is a system fully and uniquely defined. Thus following Cox and Lewis [cox66] the evolution studies undertaken by the present authors have centered on plots of evolutionary trends as a function of RSN. This also overcomes the problem that arises from the industrial practice to insert releases with decimal or other identifiers between full releases.

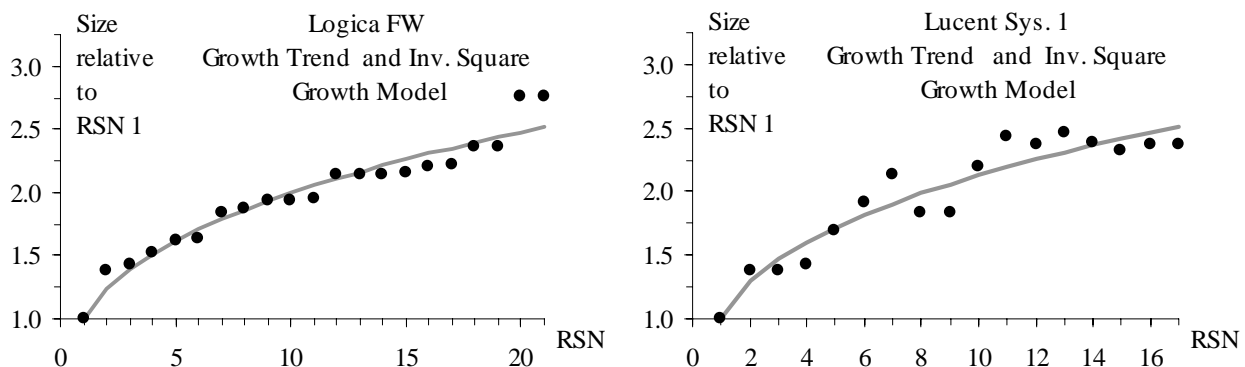


Figure 6 Growth trends of Logica FW and a Lucent system showing the inverse square growth fit

A further problem arises from the presence of minor releases. Where a release provides correction of a very minor fault, for example, its inclusion in the release sequence is not meaningful. A degree

of informed judgement must therefore be applied to select an analysable set of data from the full set. The criteria applied in practice involves many factors and must vary from system to system and even from analysis to analysis. They cannot be further discussed here [leh97a].

Real time behaviour has also been examined. But other than revealing changes in release interval policy those examinations have not changed the basic conclusions reached. Further investigation of the evolutionary behaviour in real time is, however, needed.

Comparison of the two plots in figure 6 reveals a strong similarity in long term growth and in short term ripples. More importantly, each of these trends could be fitted with an *inverse square model* that relates the sizes of the successive releases identified by the RSN sequence. The model was initially proposed by Turski [tur96]. It takes the form:

$$S_1 = y_1, \quad S_i = S_{i-1} + \hat{E}/S_{i-1}^2 \quad (i = 2, \dots, n) \quad [1]$$

where y_1 is the observed size of release RSN 1 and S_i is the predicted size at release RSN i . n is the total number of releases and the value of the parameter \hat{E} is the mean of the individual E_i calculated from, for example:

$$E_i = (y_i - y_1)/\sum_{k=1}^{i-1} (1/y_k^2) \quad (i = 2, \dots, j) \quad [2]$$

y_i is the actual size of release RSN i and j is the number of releases used to estimate the model. Other ways of estimating \hat{E} have been discussed elsewhere [leh97a].

The inverse square model provided the best fit of various alternatives attempted for the systems modelled (one in two segments) with the exception of IBM OS/360-370 where a linear fit proved more appropriate. The reasons behind this discrepancy are not fully understood. It is believed, however, that the prolonged linear growth and the increasing growth rate in moving from RSN 19 to RSN 20 may relate to the subsequent instability observed in figure 1.

The goodness of fit has been assessed by the *mean absolute percentage error* (% MAE) and the standard deviation (σ) of the percentage errors (residuals). The figures that follow are for % MAE and σ computed from data on all available releases. For the two systems, respectively, the % MAE is of order 3.6 and 6.0 and the σ is 4.8 and 7.2. Fits of similar quality have been also obtained for the systems modelled [leh98b].

The general trend shown in figure 6 and the goodness of fit of the inverse square model suggests that there is an important deterministic component in the long term growth of both these systems. The same observation may be made for the other systems analysed and also, despite the difference indicated above, for OS/360-370. This is striking when one acknowledges that demand for change tends to increase with time, that management generally seeks increased productivity and constant or increasing growth rates, that advances in technology would facilitate such increased rates and that changes in budgets, personnel, policies and organisation structure, etc could be expected to affect the growth in irregular fashion. It seems reasonable to infer that while the specific growth increment at each release is largely influenced by management decision (the selection of release content), long term trend emerges as an intrinsic property.

Without going into further detail here, interpretation of these results and those obtained from the other systems studied, suggests that the % MAE for the precision of the inverse square growth model ranges varies from 3.6% to 8% [leh98b] and, as indicated below, is essentially independent of both the number of points considered in estimating the model and the starting point of the estimation process. The software process is generally regarded as being driven, controlled and directed by humans [you98]. The remarkably close fits to a deterministic model that have been demonstrated appear to provide strong support for the FEAST hypothesis that the dynamics of

software evolution is determined by organisation, system and/or process characteristics³. Which system and which process will be considered below.

After fitting an inverse square model to the Logica FW data based on the full data set and observing the strikingly good fit, Turski asked, “over how many releases is data required to obtain an acceptable fit that yields a satisfactory predictor for future growth?” [tur96]. His result is illustrated by the lefthand plot of figure 7. The plot suggested that a surprisingly small number of points were required to to obtain a good fit. No significant additional precision is gained by computing the model beyond the knee of the plot. The lead in to the knee was interpreted as the number of releases required to establish the dynamics of the process, for the dynamics to "take over".

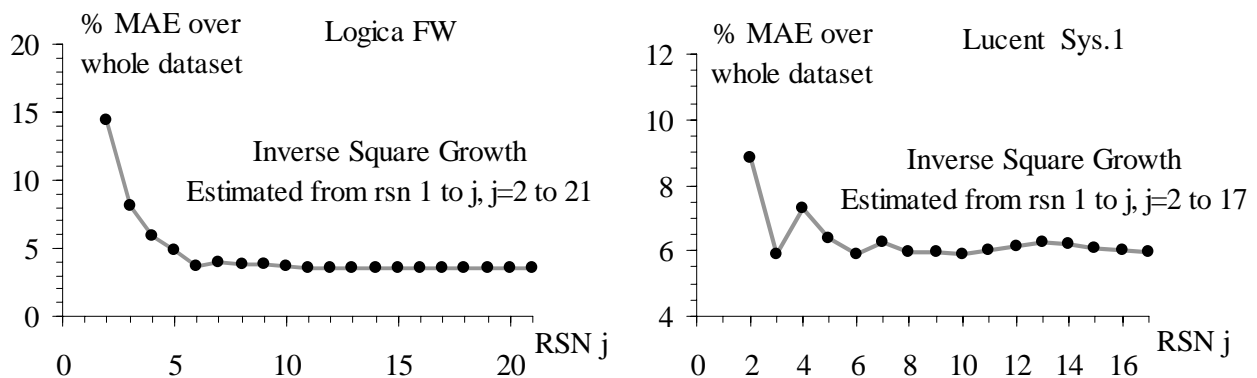


Figure 7 Mean Absolute Percentage Error of fit as a function of the number of points used to estimate the inverse square model

Until recently, this interpretation was accepted. The first doubts arose on learning that data relating to the FW system did not include early releases of that system. Thus some process and organisation properties were already established at what is here termed RSN 1. It was, however, acknowledged they may have been significantly modified by process changes known to have been adopted at that time. This led to the realisation that the knee on the left of the plot should not be primarily interpreted as the settling of the dynamics. It was far more likely to represent the convergence of the model parameter and, therefore the model, to a steady value.

To explore this question, similar plots, as shown in figure 7, were determined for the other systems under investigation. The similarity of the other plots to that for FW and, in particular, that they also appeared to converge to the effectively constant % MAE supported the tentative conclusion that it was the model (that is parameter) settling interval rather than the dynamics establishing time that was being observed. The fact that, except for changes required to estimate a linear model for OS/360-370, all the models settled to the final value in almost the same number of releases [leh98b] provided further support for the conclusion that, at the very least, model build up is an important element of the “knee phenomenon”.

It was then realised that few projects begin from scratch. Factors such as management and implementation procedures and controls, for example, all play a role in the process. Many will have been inherited from past projects and practice. For any other than a totally new organisation, general corporate practice will have been long established, formally or informally. Equally organisations adopt a generic corporate software process, that serves as a prototype for specific processes. Finally, any organisations will be subject to strong external forces, client needs, market forces and so on. All represent global forces that will directly impact a new process at even before its inception. Thereafter their impact will continue to influence events through feedback controls, checks, balances and pressure points of various forms for various purposes. That such forces are present cannot be denied.

³ Alternative interpretations cannot be discussed and evaluated here.

We thus conclude that a major, possibly dominant, component of the project dynamics in any established organisation with a prior activity in software, or more general, system development, will have been *inherited*. To examine this hypothesis a further investigation was conducted as described below and one outcome is shown in figure 8. The results, in conjunction with our other results and our understanding of the phenomenology, lead convincingly to the following hypothesis: "Long term system evolutionary growth trends are largely determined by feedback related dynamic components, many of which will have been inherited or influenced from outside the immediate technical process". The influences that determine short term behavioural patterns are still under investigation.

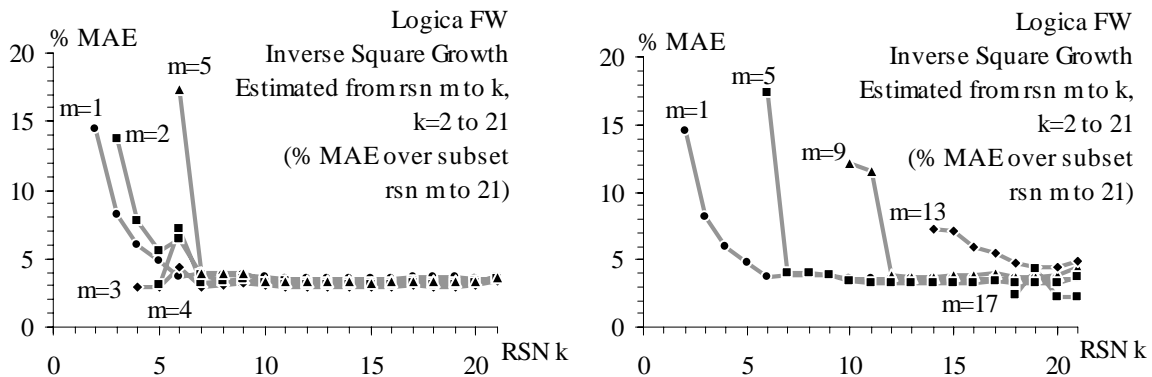


Figure 8 Initial Case for Inherited Dynamics

Plots, similar to those of figure 7, were generated again by computing \hat{E} , and hence the % MAE for models of the same systems based on varying number of points but shifting the starting point m of the sequence of data points used to both estimate and assess the model. Figure 8 provides examples of the resultant plots for the FW system. The first instance shows the values of the statistic % MAE for starting points $m = 1$ to $m = 5$, the second for $m = 1, 5, 9, 13, 17$. As is to be expected the start up behaviour of the phenomenon reflected by the plots is sensitive to initial points that show a relatively large variation from the typical values in the sequence. Deviations from the relatively smooth knee previously observed reflect such variations and do not interfere with the conclusion that the plots of figure 8 are supportive of the interpretation of the previous paragraph. Similar results were obtained for the other systems and for various variations on the experiment as described [leh98f].

It is concluded that the lead in to the knee of the plots in figure 7 reflects the process of model estimation, the convergence of its parameter to a settled value. What is more significant from the practical point of view, the observed behaviour beyond the knee is interpreted as a consequence of process and organisational dynamics, with global and inherited factors such as those identified above playing a large, probably dominant role. Further investigation is, however, required to confirm or negate this tentative conclusion.

Investigation of the inherited dynamics issue and its implications will have to await a successor project and has been included in the FEAST/2 proposal [leh98c]. One approach proposed to permit separation of the various dynamic components will be to analyse and compare amongst themselves black box growth models obtained from different sub systems in a single system, those obtained from different systems in the same development organisation and models obtained across organisations. It is also likely that an extension of the systems dynamics investigation underway in FEAST/1 [leh98a, wer98] will be able to make a significant contribution to resolution of this issue by actually identifying and modelling the mechanisms that influence behaviour dynamics from outside the immediate development project.

6 Final Remark

This paper has discussed various aspects of the role of feedback in the software process and has focused on a small subset of the recently completely FEAST/1 project results. Phenomenological reasoning, together with the observations made from software systems studied from the seventies to date, support the assertion that global software processes are and behave as feedback systems. It follows that achievement of a better intellectual control over the software process, software evolution management (and its support) and process improvement requires more than just local, technological advance. They require further progress in understanding and mastering various aspects of feedback in the global software process. Many multi-disciplinary investigations are needed, and FEAST/1 is considered just an initial, if ground breaking, attempt. The recently proposed FEAST/2 project to continue and extend these investigations awaits funding decision. Meanwhile we urge others to pick up on the explorations of these issues, of the study of the software process as a feedback system.

Acknowledgements

We are grateful to Dr Paul Wernick, who as the third member of the FEAST/1 team made many contributions to FEAST/1. We are also grateful to Professor Wlad Turski and Dr Dewayne Perry who, as Senior Visiting EPSRC Fellows, have made significant contributions to this investigation and to the underlying concepts. Grateful thanks are also due to the industrial collaborators for providing data and background information on their systems and for helping with the interpretation of the observations during work meetings in the field and periodical workshops held at Imperial College. The work described in this paper has been supported by the UK EPSRC grants GR/K86008 (FEAST/1), GR/L07437 and GR/L96561 (SVFs).

References - papers included in [leh85] are identified by an *

- [abd91] Abdel-Hamid T and Madnick SE, *Software Project Dynamics - An Integrated Approach*, Prentice Hall, Englewood Cliffs, NJ, 263 p.
- [bel72] Belady LA and Lehman MM, *An Introduction to Program Growth Dynamics*, in *Statistical Computer Performance Evaluation*, W Freiburger (ed), Academic Press, New York, 1972, pp. 503 - 511
- [cox66] Cox DR and Lewis PAW, *The Statistical Analysis of Series of Events*, Methuen, London, 1966
- [fea98] *FEAST Web Site*, Department of Computing, Imperial College, London, <http://www-dse.doc.ic.ac.uk/~mml/feast>
- [leh69]* Lehman MM, *The Programming Process*, IBM Res. Rep. RC 2722, IBM Res. Centre, Yorktown Heights, NY 10594, Sept. 1969
- [leh74]* *id.*, *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 for 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*, J. of Sys. and Software, v. 1, n. 3, 1980, pp. 213 - 221
- [leh80b]* *id.*, *Programs, Life Cycles and Laws of Software Evolution*, Proc. IEEE Special Issue on Software Engineering, v. 68, n. 9, Sept. 1980, pp. 1060 - 1076
- [leh84] Lehman, MM, Stenning V, and Turski WM, *Another Look at Software Design Methodology*, Res. Rep. 83/13, Dept. of Comp., Imp. Col., London, June 1983. Also, *Software Engineering Notes*, v. 9, no 2, April 1984, pp. 38 - 53
- [leh85] Lehman MM and Belady LA, *Program Evolution, - Processes of Software Change*, Acad. Press, London, 1985, 538 p.
- [leh89a] Lehman MM, *Uncertainty in Computer Application and its Control Through the Engineering of Software*, J. of Software Maintenance: Research and Practice, v. 1, n. 1, Sept. 1989, pp. 3 - 27
- [leh89b] Lehman MM, *Uncertainty in Computer Application*, CACM, v. 33, n. 5, May 1990, pp. 584 - 586
- [leh94] Lehman MM, *Feedback in the Software Evolution Process*, Keynote Address, CSR Eleventh Annual Workshop on Software Evolution: Models and Metrics, Dublin, 7 - 9th Sept. 1994, Workshop Proc., Information and Software Technology, sp. is. on Software Maintenance, v. 38, n. 11, 1996, Elsevier, 1996, pp. 681 - 686
- [leh94,95] Lehman MM (ed.), *Pre-prints of the Three FEAST Workshops*, Dept. of Comp., Imp. Col., London, 1994/1995, available via <http://www-dse.doc.ic.ac.uk/~mml/feast/papers.html>
- [leh96a] Lehman MM, *Laws of Software Evolution Revisited*, Proc. EWSPT'96, Nancy, 9 - 11 Oct. 1996, LNCS 1149, Springer Verlag, 1997, pp. 108 - 124
- [leh96b] Lehman MM, Perry DE and Turski WM, *Why is it so hard to find Feedback Control in Software Processes?*, Invited Talk, Proc. of the 19th Australasian Comp. Sc. Conf., Melbourne, Australia, 31 Jan - Feb 2 1996. pp. 107-115.
- [leh96c] Lehman MM and Stenning V, *FEAST/1: Case for Support*, Dept. of Comp., Imp. Col., London, March 1996

- [leh96d] Lehman MM, *Process Improvement - The Way Forward*, Invited Keynote Address, Proc. Brazilian Software Engineering Conference, 14 - 18 October 1996, pp. 23 - 35
- [leh97a] Lehman MM, Perry DE, Ramil JF, Turski WM and Wernick PD, *Metrics and Laws of Software Evolution - The Nineties View*, Proc. Metrics '97, Albuquerque, NM, 5 - 7 Nov., 1997. Also as *Process Improvement - The Way Forward*, in *Elements of Software Process Assessment and Improvement*, IEEE CS Press, 1998
- [leh97b] Lehman MM, *Process Models - Where Next?*, "Most Influential Paper of ICSE 9 Award", Proc. ICSE 19, Boston, 20 - 22 May 1997, pp. 549 - 552
- [leh98a] Lehman MM. and Wernick P, *System Dynamics Models of Software Evolution Processes*, Proc. Int. Wrksh. on the Principles of Software Evolution, ICSE'98, Kyoto, Japan, April 20 - 21, 1998, pp. 6 - 10
- [leh98b] Lehman MM and Ramil JF, *The Impact of Feedback in the Global Software Process*, ProSim'98, Proc. Int. Wrkshp on Softw. Proc. Simulation Modelling, June 22 - 24, 1998, Silver Falls, Oregon, also to appear in *Journal of Systems and Software*, 1999
- [leh98c] Lehman MM., *FEAST/2: Case for Support*, Dept. of Comp., Imp. Col., London, July 1998
- [leh98d] Lehman MM, Perry DE and Ramil JF, *Implications of Evolution Metrics on Software Maintenance*, Proc. of Fifth Int. Int. Conf. on Softw. Maintenance, ICSM '98, Bethesda, Maryland, Nov. 16-20, 1998
- [leh98e] Lehman MM, Perry DE and Ramil JF, *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
- [leh98f] Lehman MM, Ramil JF and Wernick P, *The Influence of Global Factors On Software System Evolution*, Res. Rep. 98/11, Dept. of Comp., Imp. Col., London, October 1998
- [mad96] Madachy RJ, *System Dynamics Modelling of an Inspection Process*, Proc. 18th Int. Conf. on Softw. Eng., Berlin, 25 - 29 March 1996, IEEE Comp. Soc. ord. n. PR07246, IEEE Cat. n. 96CB35918, pp. 376 - 386
- [tur96] Turski WM, *Reference Model for Smooth Growth of Software Systems*, IEEE Trans. on Soft. Eng. v.22, n.8, Aug. 1996
- [ven95] *Vensim Reference Manual*; Version 1.62; Ventana Systems Inc.; Belmont, MA; 1995
- [wer98] Wernick P and Lehman MM, *Software Process White Box Modelling for FEAST/1*, ProSim'98, Proc. Int. Wrkshp on Softw. Proc. Simulation Modelling, June 22 - 24, 1998, Silver Falls, Oregon, also to appear in *Journal of Systems and Software*, 1999
- [you98] Yourdon E, *A Tale of Two Futures*, IEEE Software, Jan./Feb. 1998, pp. 23 - 29