

# Process Improvement - The Way Forward

M M Lehman  
Department of Computing  
Imperial College  
London SW7 2BZ  
tel: +44 (0)171 594 8214  
fax: +44 (0)171 594 8215  
mml@doc.ic.ac.uk

## Abstract

The paper briefly reviews the difficulties encountered in achieving further major improvement in some aspects of the software evolution process. This may, it is suggested, be due, in part, to the fact that the global process is a complex, multi loop multilevel feedback system. After summarising the results of a 1970s study and the laws of software evolution that arose from it, the paper introduces a FEAST hypothesis. This asserts that the global software process for *E*-type systems is such a multi loop feedback system and displays the properties associated with such systems. Early results from the FEAST study culminating in the FEAST/1 project started in October 1996 are outlined. The support these results provide for the original five Laws of Software Evolution and for three extensions are also indicated. The FEAST/1 project is to explore the phenomenon in depth through modelling of a number of industrial projects and will seek to demonstrate its impact. The early work of the 1970s concentrating primarily on OS/360 as confirmed by the FEAST results obtained to date suggest that mastery of the process as a feedback system is key to further process improvement.

## 1 Software Process Improvement

### 1.1 Overview

In their early years digital computers were primarily regarded as tools for the automation of numerical computation. In the subsequent, now over forty, years of evolution of digital computing, applications and their operational domains have been extended in their variety, in the detail in which they are addressed and in the extent to which the computer system is integrated with and into the application, its operational environment and the activities of the humans in that environment. This growth is reflected in the range of functionality of current operational systems. The size, functional complexity and structural complexity of the software developed and successfully used has increased by several orders of magnitude. This is reflected in an even greater increase in the number, variety and operational complexity of the features incorporated in the software. The operational domain, the application, activities in that domain and the characteristics of the humans and mechanisms involved are reflected in the software in ever greater detail. The intimate coupling between the software, the operational environment and the humans in that environment as reflected in the growing complexity of the interface.

The search for improvement in the software evolution process<sup>1</sup> can be traced back to the very beginning of digital computers and of programming. In the preface to their book *The Preparation of Programs for an Electronic Digital Computer*, [wil51] Wilkes, Wheeler and Gill wrote, in connection with their invention of the concept of subroutines, "The methods of preparing programs for the EDSAC described in this book were developed *with a view to reducing to a minimum* the amount of *labour* required and hence of making it *feasible* to use the machine for problems that require only a few hours of computing time as well as for those which require many hours ..."<sup>2</sup> The search for improvement has been in the forefront of programming research and development ever since, though the term *process improvement* has only come into common usage in the last decade. Innovations relating to attributes such as programming productivity, product quality, process predictability and responsiveness indicate the success of that search. They include high level languages, structured programming, abstract data types, new programming paradigms, formal methods, metrics, CASE, support environments, process modelling, model based process improvement and so on. The list is endless.

<sup>1</sup> At the highest level of detail processes of software development and maintenance are equivalent sharing a common underlying paradigm [leh84]. Each, in its own way, achieves system evolution. Hence, unless otherwise stated, the term *software evolution process* or *process* in short as used throughout this paper includes both activities.

<sup>2</sup> This author's italics

There have, unfortunately, been few studies that provide convincing quantitative data on the actual benefit that such innovations yield. The awesome increase since the early days of computing in functional complexity and power of software systems being developed, used and maintained testifies to the fact that major progress has been made. So does the fact that development times and software costs have, in many instances, decreased significantly. But maintenance activity remains as intensive as ever, time to delivery and the ultimate cost of software development projects is still not reliably predictable, functionality and quality of early releases of a system are below specification, often below par. There continue to be too many cases of unsatisfactory or even incorrect system behaviour after installation, too many disastrous failures, too many projects that never deliver.

This is not to suggest that individual process innovations did not represent a technological advance. They clearly yielded local improvement in one way or another, that is at the point of application. Use of high level languages, for example, led to increases in the average rate of code generation of order five or so. Further significant benefit resulted from the increased understandability that the use of higher level languages yields and the consequent improvement in the quality, structure, maintainability and evolvability of code. These aspects led to significant reductions in development time, reduced need for fault fixing maintenance and so to growth in productivity over system lifetime. But their adoption has not solved the overall *software problem*. as indicated in the previous paragraph. Formal methods have prospered in academic research. They provide the rationale and a methodological base for many CASE tools. But their contribution to industrial effectiveness is limited. CASE tools yield local benefit to their users through the methods they make available and by the rigour they impose. Yet they have not made a major impact on industrial software development effectiveness. And so it goes on.

Introduction of such innovations has generally been based on the argument that their general adoption by industry will, once and for all, end the software crisis. Failure to live up to this expectation can be rationalised and individually explained. That path is not pursued in the present paper. The fact remains that general experience over innovations, over organisations, over different process structures and application areas is that global benefit derived from the introduction of an individual improvement is much more limited than might be expected from its impact at the local level. This suggests that it will be more profitable to seek a common explanation. It would seem likely that some intrinsic property of the process is constraining improvement of the global process followed to develop or adapt an operational computer system to satisfactorily address market needs.

The most obvious explanation is that expectations of innovators are often unreasonable. But that hardly explains the slow progress in achieving major improvement in overall effectiveness of the total process from conception to system operation and its subsequent evolution. Clearly also the introduction of individual languages, methods, tools, paradigms or programming steps or activities impacts only a small portion of the total effort required to take a system from conception to deployment in the field. Consider, for example, **doubling** the effectiveness, that is halving the duration, of an individual activity through the introduction of a new or improved language, method or tool. In a primitive process that step might have originally absorbed 50% of the total process effort - work expended or time required - to take the product from conception to installation in the field. The total process effort or duration is therefore reduced by 33%, a visible and significant improvement. In today's more sophisticated industrial processes few, if any, steps represent more than, say, 5% of the total effort from start to finish. Thus the externally visible impact of the same improvement would be of order well under 3%, a trivial improvement relative to the 50% local gain. Indeed it would probably be lost in the noise, the variations that occur from process to process due to variations in environmental, including project, conditions. If improvement is measured in terms of quality a similar argument applies. The impact of the quality achieved in individual steps on global quality is likely to vary considerably. The contributions of individual steps to achieving a quality product and on the effective productivity of individual steps is likely to vary widely between steps but will also be much smaller at the global level than locally.

Other explanations are also possible but will not be discussed here [leh95]. Instead, attention is drawn to a constraint which, with hindsight, is almost self evident. The phenomenon from which underlies it was first identified over 25 years ago but only recently has the penny dropped.

## 1.2 Recent Developments in the Search for Process Improvement

The very first recognition of and advance in process improvement was almost certainly in the 1951 book by Wilkes *et al* as quoted above, referring to their "invention" of the concept and management of *subroutines* in the context of the Cambridge University EDSAC development [wil51].

Until recently the approach to further improvement was indirect. Research and development interest concentrated, in general, on specific artifacts of or activities in the software *development* process. Process maintenance received scant attention. The early focus was on procedural programming languages, considering both level and style. With regard to the former, languages have advanced from binary codes and machine languages, through assembly languages to a large variety of (so called) high and very high level languages. Alternative paradigms have also been explored. In general, however, procedural programming has not been displaced in primary industrial usage other approaches, functional programming via transformation or logic programming, for example, have been widely explored and applied. Over the years concepts relating to the syntax, semantics and use of programming languages have emerged. Examples include structured programming [dij68, knu74, you79], successive refinement [wir71] or, more recently, object orientation [boo86]. Together these led the way to more detailed consideration of the overall technical programming process.

It was, for example, successively recognised over many years that program development must be preceded by a design activity, that this must be preceded by development of a specification and that this, in turn, must be based on a requirements statement which itself had been derived from a requirements analysis. More recently the need for an initial application domain analysis has been recognised though this may well be just another term for what had previously been termed *systems analysis*.. Activities to implement these needs were then developed, initially on an *ad hoc* basis, and added to, in general, a *waterfall* [roy70, boe76] type process.

And so with other aspects of the software process. The need to understand and improve it was recognised from the beginning of the digital computer age [wil51, ben56, leh69, roy70, boe76]. The discussion at the NATO Software Engineering conferences [nau69], [bux70] should also not be overlooked. Wider awareness was triggered by a series of International Process Workshops, the first of which was held in 1984 [spw84]. A keynote lecture by Osterweil [ost86,97] and a response to thereto [leh86,97] triggered strong interest in modelling the process, particularly in academia. The primary focus of the modelling effort was on understanding the process as such. The search for further significant improvement was, however, never far away. More conscious and directed work on process improvement was a direct outcome of research and development at the Software Engineering Institute (SEI) at Carnegie Mellon University. This culminated in their CMM models, related improvement technology and the emergence of an international SPIN (Software Process Improvement Network) movement.

## 2. The Laws of Software Evolution

A 1968/9 study of the IBM programming process [leh69]<sup>3</sup> provided, *inter alia*, quantitative data on various attributes relating to successive releases of IBM's operating system OS/360 and on the effort expended in going from release to release. As data on further releases became available a series of models reflecting the growth trends of the system and attributes of the process whereby it was being evolved [leh85] were developed. Their analysis led, in turn, to a descriptive phenomenology. This suggested that behaviour reflected in the observations was primarily due to the influences of human and organisational factors. That is, the characteristics of system evolution were largely determined by factors other than the process and technology being used to achieve that evolution. This observation and its implications were summarised and encapsulated in a series of *behavioural* statements, that were exogenous to the technology being used in the evolution process. From the point of view of the software engineer they were therefore to be viewed as *laws*..

The first three of these laws were formulated in the mid seventies [leh74] and discussed in greater detail in 1978 [leh78]. Two further laws were introduced in 1980 [leh80a]. A sixth was introduced in a subsequent footnote [leh91]. The remaining two while publicly discussed have only recently been

---

<sup>3</sup> References identified by a \* in the listing are reprinted in [leh85].

published [leh96c]. As restated<sup>4</sup> explicitly below the laws relate to *E*-type systems [leh80b] that is, broadly speaking, to software systems that solve a problem or implement a computer application in the *real world*.

No.	Brief Name	Law
I 1974	Continuing Change	An <i>E</i> -type program that is used must be continually adapted else it becomes progressively less satisfactory
II 1974	Increasing Complexity	As an <i>E</i> -type program evolves its complexity increases unless work is done to maintain or reduce it
III 1974	Self Regulation	The <i>E</i> -type program evolution process is self regulating. Distribution of product and process attribute measures are close to <i>normal</i>
IV 1980	Conservation of Organisational Stability (invariant work rate)	The average effective global activity rate on an evolving <i>E</i> -type system is invariant over the product lifetime
V 1980	Conservation of Familiarity	During the active life of an evolving <i>E</i> -type program the average content of successive releases is invariant
VII 1980	Continuing Growth	The functional content of an <i>E</i> -type program must be continually increased to maintain user satisfaction over its lifetime
VII 1996	Declining Quality	<i>E</i> -type programs will be perceived as of declining quality unless rigorously maintained and adapted to a changing operational environment
VIII 1996	Feedback System	<i>E</i> -type programming processes constitute multilevel, multi loop, feedback systems and must be treated as such to achieve significant improvement over any reasonable base

**Table 1. Laws of Software Evolution**

A recent review [leh96b] extends the earlier [leh78,80] discussion. The present paper is oriented to software process improvement and, therefore, addresses the laws only to the extent relevant in that context. The eighth law, in particular, is central to its principal thesis since it states the fact that the *E*-type evolution process is intrinsically a feedback system. As is shown below, effective change of that process, with whatever objective, requires understanding of the feedback characteristics and the means whereby these can, and normally must, be adjusted to achieve the desired objective. As new data is obtained, is shown to be consistent with the laws and provides further understanding of and insight into software process phenomenology, it suggests ever more clearly that the other seven laws are a consequence of the eighth law, that is the feedback property or, at least, closely linked to it.

### 3 Software Evolution

The 1970s study included the feedback property 1972 [bel72]. Discussing an earlier version of the OS/360 IBM operating system growth curve reproduced in figure 1, it was observed that, "... the ripple is typical of a *self stabilising process* with positive and negative feedback loops. From the long-term point of view 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 ...".

As observed in the quotation above the ripple in this plot suggested that negative feedback may be an indicator of self regulation of OS/360 evolution. The *chaotic* (in a strictly technical sense) behaviour exhibited over the final releases is, similarly, likely to have been due to feedback. It appears to reflect *instability* resulting from the positive feedback that led to pressure for excessive functional growth in evolving release 20 from release 19.

In summarising the observations to that time it was subsequently suggested [leh78] that one should "... 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 ...". And that was how the matter was left.

<sup>4</sup> Numbered in order of formulation and publication. Over the years the names and wording of the laws have been modified. The fundamental understanding they reflect remains, however, essentially the same although the focus of attention may have shifted.

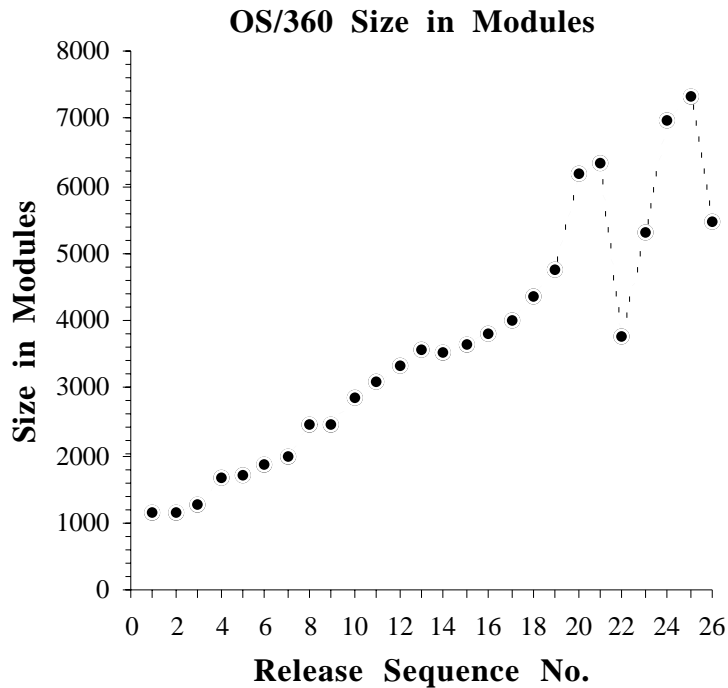


Figure 1. The growth of OS/360

#### 4 The Search for Software Process Improvement

As pointed out in section 1, the search for improvement of the constituent parts of the software process is almost as old as digital computing itself. Early interest in process improvement, and by implication, in the software process led to widespread development and application of software process models and meta-models. The later were perceived as useful for, for example, driving project support environments and for the study, control and improvement of the process improvement process. These have now led to disciplined procedures for process improvement spearheaded by the Software Engineering Institute of Carnegie Mellon-University. The CMM [pau93] and other models now provide the foundation for world wide activity as typified by the international Software Process Improvement network. It seeks to improve the industrial process of software development and evolution (maintenance) by one or more orders of magnitude [maj93].

Little, if any, real, quantitative assessment has been undertaken of the effectiveness of the various approaches taken to process improvement or of the tools that support them. As is normal in computer science and software engineering, each of them has its ardent advocates. On the other hand there are those who have serious concerns and reservations [gra96]. It is, however, apparent that much of the improvement efforts to date has concentrated on what the CMM model classifies as levels zero, one and, to a lesser extent, two. Thus the successes reported are also from and to these lower levels. This is, of course, not surprising. For one thing, the majority of industrial development processes operate at these low levels. Even where an individual groups or sub organisations have succeeded in improving their process to level three or above, processes in the remainder of the organisation will, at best, be striving to emulate them. Moreover, it is more straight forward to improve primitive processes and, seemingly also more important in terms of business effectiveness and success. Large corporations and those developing major systems, particularly such as operate in the real time and safety critical areas, have, on the other hand much higher stakes, including human lives. Many such organisations are, therefore making major investment in the search for software improvement.

Interestingly, there may be an additional reason, why major success in process improvement appears, for the moment, to have been concentrated at the lower end of the process effectiveness scale. The realisation that the software process constitutes a multi loop, multi loop, multi agent feedback system brings with it a fundamental property to be developed and briefly discussed in the remaining sections of this paper. The FEAST hypothesis to be introduced in the next section includes the observation that, in general, the externally visible behaviour and performance of a

feedback system is unlikely to be significantly changed by changes to the forward path alone. The feedback paths and their mechanisms must also be adapted to permit full benefit, or even any benefit at all, being derived from forward path changes.

This leads to the question already raised "Why, despite the major effort and enormous resources invested worldwide in the development of languages, methods, support tools and, most recently, process improvement and despite the many resultant innovations in methodology [gri78], process organisation, process support and project management does industrial software systems development still suffer major problems? Why, except for primitive processes, is it so difficult to achieve significant improvement in global process attributes?" It is now suggested that it is due to the fact current approaches have, in general, not taken cognizance of the feedback nature of the global process system, as expressed, for example, in the eighth law. The failure of individual innovations as previously identified [leh95,6c]. to yield significant improvements at the global process level, suggests that this is likely to be a direct consequence of the feedback loops that engulf them and the feedback mechanisms that control them. This topic is the core of an ongoing (1996 - 8) project FEAST/1 (*Feedback, Evolution And Software Technology*) project as outlined below.

## 5 FEAST

### 5.1 Feedback Systems

It is undeniable that the global *E*-type software evolution process contains a large number of feedback loops. Provide for and result in information sharing and control, some formal some informal. They exploit both formal and informal communication links and control mechanisms. Together the global aggregate provides the basis of an *E*-type process that constitutes a complex feedback system. The feedback involves project management, technical development, quality assurance, system validation, process design, its support and control, user support and many other activities. It also involves many levels of corporate personnel including corporate executives, marketeers and so on. Equally, it involves, each at the appropriate level of interaction, organisational and individual users. The direction, quality, effectiveness and output of the process is a complex function of the directives, control and information flow between these many *agents* and between the *agencies* within which they work. This information flow drives and guides the process, seeking to motivate, direct and control it. Positive feedback signals trigger or accelerate growth. They may lead to instability. Negative feedback, on the other hand, stabilises. When applied over some forward path element, be it a single mechanism or a subsystem, changes in its output in response to changes in its characteristics or makeup are reduced by approximately the gain in the feedback loop. The precise impact depends, of course, also on the delay or phase shift in the loop. By applying negative feedback, checks and balances for example, one achieves global stability in the face of changes in the characteristics of individual elements.

By analogy with other feedback systems and from an understanding of their characteristics and meta behaviour it should be expected that the software process has characteristics such as the following. With many feedback paths in a system, the relationship between internal changes and observable external behaviour is thus most complex. Characteristics of communication paths between elements, of control paths and of internal interactions all influence that relationship. Replacement of an element with one having different characteristics, even addition of a new element, may, therefore, make no significant or even perceptible difference to observed global behaviour outside the immediate region where the change was introduced. Changes to forward path elements alone have a far smaller global impact than analysis of the local impact would suggest and is likely to be small in relation to the magnitude of the elemental change. As suggested in the previous section, adjustment of the feedback paths and/or mechanisms that *contain* system elements being changed is required if the local changes are to have global impact. The precise nature of changes required is neither simple nor self evident.

These facts more than suffice to explain the resistance of the global software process to change. They represent and explain precisely what is observed in the software evolution process, supporting the assertion derived from structural analysis that the process constitutes a feedback system. As such it must, therefore, be expected to display basic feedback system properties. The impact of internal changes to process mechanisms will inevitably be constrained by the feedback paths in the process

and the organisations within it is embedded and applied. The visible benefit derived from the introduction of improved languages, methods, procedures or tools in the forward, development path of the process can only have limited impact. Locally an innovation, whatever its nature, might prove to be most beneficial, yielding significant improvement in productivity, quality, responsiveness or whatever. But such gain will, generally, be attenuated or even negated by the feedback mechanisms that certainly modify, and may largely determine, overall process characteristics.

## 5.2 Process Improvement

Once it has been accepted that the software process is a feedback system, it becomes obvious that efforts directed to its improvement cannot focus exclusively on forward path technical development. Nor is it sufficient to extend consideration to definitional steps such as requirements engineering or system specification and design. Communication channels between all such technical development activities, between them and the organisations within which they are embedded and between both these and the user community they seek to serve, must be considered. These channels include feed forward and feedback control mechanisms. It is their joint operation that determines the characteristics of the global process and it is improvement of this total process which brings the real gain. To achieve such improvement and to seek maximisation of the benefit, requires tuning of the total system.

Improvement efforts must consider development and control influences stemming from all organisational levels. Executive management sets global goals. Local management sets operational objectives. As a result of changes in the economic climate, in business needs, in the development and operational environments, long term goals, immediate targets and even strategies are changed. Marketing, sales and user support feed back information and requests that impact the process. Software engineering activity that defines and controls the process, its support and progress of the product through its various stages of development exercise significant influence. All impact product and process goals, product distribution, installation and introduction into usage and the details of the product evolution process. Their reactions and influence must be taken into account when process changes are proposed. Much of that influence is achieved via feedback, whether in the form of control or as information that influences local decision taking.

Software products are never an end in themselves. They are a means to an end. The real benefit accrues from the use of the software. This benefits development and client organisations, the user community and beyond. All exert pressure via feedback, each seeking to maximise the benefit as measured in its own terms. For a system as complex as the software process the feedback inevitably impacts system, that is process, behaviour in unintended and unforeseen ways. Elimination of this uncertainty requires mastery of the process dynamics to permit control of its behaviour.

## 5.3 The FEAST Hypothesis

The ignoring of feedback in the software improvement process explains why major improvement has been difficult to achieve. It, must however, remain a hypothesis until supporting evidence is obtained or until proven false. A study of industrial processes can provide such evidence through identification of loops, their inclusion in process models and identification of the constraints they impose. Doing this opens the way to methods for representing, modifying and exploiting process dynamics, a new approach to process improvement and prospects for new support tools. The hypothesis will also become the basis for a theory of the software process and of software product and process evolution. Achievement of even a part of all this would constitute significant progress.

The seed for such a study has been formalised in a hypothesis named the FEAST hypothesis. It expresses the observations and insights outlined above. It has developed from the 1970s observation, their reflection in the laws outlined above and on more recent insights. It has been widely presented and discussed. As a result its formulation has changed somewhat since first proposed [fea94]. Other versions appear in [fea94/5], [leh95,96b]. The most recent follows.

### **Hypothesis**

- As complex feedback systems, *E*-type software processes evolve strong system dynamics and with it the global stability characteristics of other feedback systems. Consequent stabilisation effects are likely to constrain efforts at process improvement.

### **Supporting observation**

- Industrial processes evolve, developing, *inter alia*, positive feedback to drive organisational progress and growth, and negative feedback controls to provide checks and balances.

The hypothesis implies three assertions:

- I The software evolution process for *E*-type systems is a complex feedback system.
- II However effective forward path changes to the process may appear *locally*, negative feedback will constrain the *global* benefit they yield. (Appropriate positive feedback may increase the benefit but must be meticulously controlled to avoid instability.
- III Major improvement to the process requires the knowledge and ability to change its dynamics by modification and tuning of its feedback structure and mechanisms.

A lemma also follows:

- Slow progress in process improvement may be due, at least in part, to lack of attention to feedback phenomena.

## **6 FEAST/1**

The FEAST hypothesis and the phenomenological analysis from which it was derived have their roots in the original programming process study [leh69]. This was strengthened and extended by subsequent observations of OS/360 and other systems as summarised in section 3 above and as discussed over the years in greater detail in the quoted references. More recent pondering about the apparent difficulty in achieving process improvement led to identification of the process as a feedback system, to recognition of the resistance to change that this implies and, hence, to the formal hypothesis statement. It has now triggered a systematic investigation, FEAST/1, intended to confirm the validity and relevance of the hypothesis and develop means for its exploitation. The study should also provide a theoretical base and framework for process improvement, theory that is long overdue.

Three workshops [fea94,5] held at Imperial College with international participation over the past two years have laid the foundations for such an investigation. Their main goal was to expose the hypothesis and underlying concepts to a wide forum, to arouse interest and to lay foundations for one or more international collaborative projects that would conclusively address what is recognised as a difficult, multi-disciplinary, challenge.

The ground work for these workshops and for FEAST/1 was undertaken by a core group consisting of Professors V Stenning and W M Turski, Dr D E Perry and the present author. This group has been meeting at intervals over the last two years to clarify the hypothesis, the basic concepts that underlie it and the nature of an investigation needed to demonstrate its validity, significance and potential. Their discussion concentrated on issues such as the meaning of *positive* and *negative* feedback in the software process context, differences between control and information feedback, a search for examples of the phenomenon and how it might be investigated [leh96a]. Their conclusions included hypothetical instances of feedback based on their understanding of typical industrial processes. Examples based on real experience in actual processes proved more illusive. The study of industrial projects is, clearly, of high priority in any systematic investigation.

A proposal was prepared [leh96b] and the two year FEAST/1 project is now funded by EPSRC under grants numbers GR/K86008 and GR/L07437. It will commence in the Autumn of 1996 with Professors B Rustem, V Stenning and the present author as Principle Investigators. The industrial collaborators are BAe, ICL, Logica and MOD. These organisations will provide access to real software development and evolution processes for study. Logica has indeed done so even before the project has formally commenced. The full study will involve observation of several of the collaborators' projects, identification of feedback paths and mechanisms, the construction of black box models that identify growth and other trends, white box models of the process dynamics and other models of system behaviour. Together these will permit assessment of the impact of feedback and any constraints it imposes on the benefit to be derived from improvement to elements of the processes studied. This, in turn, must lead to proposals for process improvements that can first be tried out and evaluated in the collaborators' environments and then generalised.

The overall goal of FEAST/1 is to demonstrate that adoption and exploitation of the feedback perspective significantly enhances the ability to improve the software process. Subgoals will:

- (a) provide objective evidence that feedback phenomena and the consequent systems dynamics have substantial impact in the software process,
- (b) demonstrate that the phenomena can be exploited both for the management of industrial processes and for their improvement,
- (c) produce justification for a more substantial, interdisciplinary, study based on the feedback perspective.

In addressing these goals specific objectives are to:

- (i) provide models of the long term (multi-release) behaviour of industrial software processes and their products that are more extensive and precise than any hitherto available,
- (ii) provide calibrated and validated system dynamics models of development processes for individual software releases,
- (iii) use these models to expose feedback phenomena and their impact within such processes,
- (iv) demonstrate the predictive capability of the models,
- (v) identify process improvements to the collaborators' processes suggested by the models and assess their impact in practice,
- (vi) demonstrate the need for further studies and define their nature.

## 7 Information Feedback and Feedback Control

The previous discussion has talked glibly of feedback with only brief reference to the very different concepts of *information* and *control* feedback. The former may or may not be absorbed by each recipient, may be correctly or incorrectly interpreted and may be acted upon or ignored. Analysis of its role and consequences is a matter for the various soft disciplines that address human and organisational behaviour, psychology, management science, organisation theory and so on. Person to person information flow is clearly not amenable to rigorous analysis. Psychologists might care to attach different probabilities to various reactions. The software engineer should not be expected to provide a meaningful model. Collective information flow in the global process should, on the other hand, be more amenable to analysis. Despite the fact one cannot predict precisely the decisions or actions to be taken by each individual one may assume that information received will, somehow, be taken into account. Thus with many individuals taking numerous decisions, data that reflects the impact on the process as a whole is, in some sense, *normally* distributed. This was, in fact, precisely the phenomenological interpretation that resulted in formulation of the third law [leh78,80a]. It was shown to be the case in, at least, one instance [cho81].

It is concluded that *information feedback*, though clearly relevant to the overall FEAST investigation, requires an interdisciplinary approach for which the time is not yet ripe. The legitimacy of the hypothesis, its implications and its exploitability must first be demonstrated to generate support for such an approach. In the interim, human observation, action and reaction must be accepted as an integral part of and influence on organisational behaviour and, in particular, on the system dynamics. It must therefore be reflected in models of the global process. It is currently assumed that in FEAST/1 it will be included through statistical distributions and/or stochastic inputs.

On the basis of results already obtained and of work in other areas the immediate prospects for successful systematic and rigorous analysis of *control feedback* in the software process are more realistic. Such analyses have, after all, been undertaken for many years in, for example, control theory applied to economic modelling [bec94] and, more recently, in studying and seeking to improve business processes [sen90]. Moreover, the pioneering work of Abdel Hamid and Madnick in modelling and analysing the dynamics of software project management indicates [abd91] that such techniques may be usefully applied in the software process area even if communication flow, forward and backward, cannot be precisely or predictably modelled at the individual or small group level. Clear understanding of the dynamics that controls an active software process, will open up new approaches to significant process improvement. That this is likely to happen was suggested by the 1970s data and has now been confirmed by results obtained in preparation for FEAST/1.

## 8 Preliminary Results

Preliminary results from the analysis of a Logica plc banking transaction system, FS, are already available. This 5 year old system installed on some one hundred user sites has seen over twenty documented releases and sub releases. Data records start at release 3 and extend over a total of

twenty one release sequence numbers (rsn). Note that, as discussed in the above publications, software process black box analyses such as that to be briefly outlined here are usefully based on a pseudo time variable such as rsn. This permits consistent consideration of releases and sub releases in the system evolution path. The use of the rsn is also more meaningful in this black box context because the moment of release represents a point of functional, design and textual system *stability* in an otherwise continuously changing environment. This black box analysis is similarly, based on *module* rather than the more usual lines of code (locs) count. Module counts are more rational because modules have more functional integrity in the context of system behaviour than do individual locs. More meaning can, therefore, be attached to them. The OS/360 investigations did demonstrate that analyses based on calendar time and locs produced results not significantly different from those that are rsn and module based. The latter are simply smoother.

Figure 2 shows the growth of FW in modules as a function of the release sequence number.

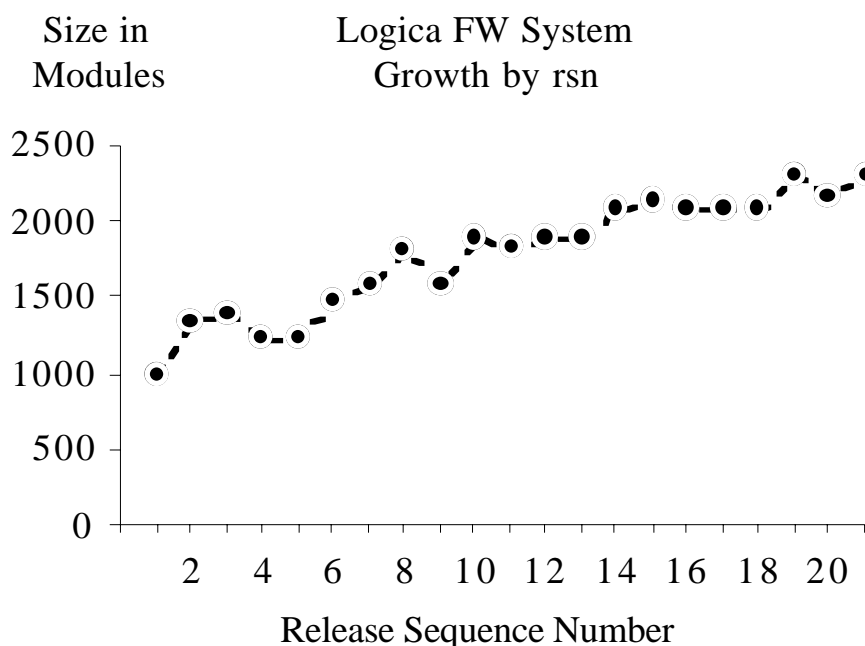


Figure 2. Growth of the FW System

This growth curve is reminiscent of that of OS/360 shown in figure 1 in at least two respects. It clearly supports the sixth law of Continuing Growth. Furthermore it also displays the ripple effect that first suggested the self stabilising behaviour of the software process, though it is far more pronounced than that for OS/360. FW therefore provides further evidence in support of the third law, Self Regulation and the eighth law which identifies the software process as a feedback system. These represent, of course, different aspects of the same underlying phenomenon. Finally, as a non IBM, non operating system, evolved using 1990s process technology FW also negates the 1970s suggestion that inferences from OS/360 are not generally applicable [leh96c].

Turski [tur96] has investigated a number of different fits to the data plotted in figure 1. He has shown that an inverse square relationship provides a remarkably good model of the main trend on which the ripple is superimposed. Given  $S_i$  as the size in modules of the release with sequence number "i", his inverse square growth law states that:

$$S_{i+1} = S_i + \frac{E}{S_i^2} \quad (1 \leq i \leq 20) \quad [1]$$

$E$  is an *effort* constant that represents the work done (in unidentified units) to take the system from one release to the next. It is the average of individual  $E_i$  computed across the set of observed system size from  $S_1$  to  $S_{21}$  as in equation [2] below. The inverse square relationship is supportive of the second law, Increasing Complexity and is typical of a system whose behaviour over time is dominated by its system dynamics.

An indication of the closeness of the fit is provided by figure 3. Note that although equation [1] does not (yet) include a ripple component, the latter settles down to an amplitude less than 10% of system size and approaching constancy. This close fit with a constant parameter  $\underline{E}$  provides further support for the third law. More generally, the data plotted in figures 2 and 3 provides support for several of the laws as summarised in table 2 in section 9.

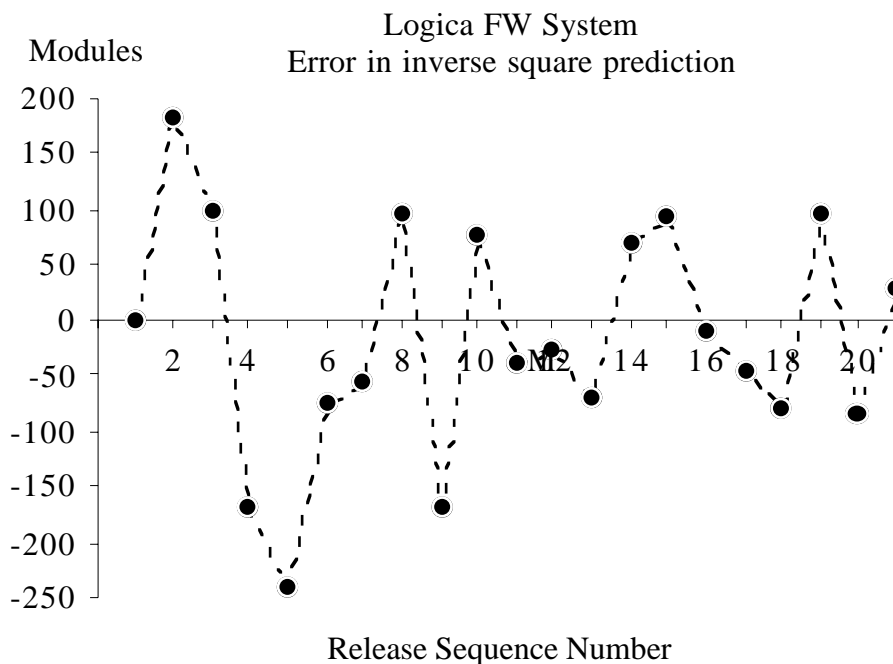


Figure 3. Difference between actual size and inverse square size prediction for FW

For the above plot, the constant "effort" parameter  $\underline{E}$  has been computed as the average of the 20  $E_i$  obtained by applying equation [2] as derived from [1] to the 21 data points from rsn 1 to rsn 21.

$$E_i = (S_{i+1} - S_i) / \sum_{j=1}^i (1/S_j^2) \quad (1 \leq i \leq 20) \quad [2]$$

Figure 3 suggests that the first 5 or so points are *outliers* that result in larger differences between the inverse square prediction based on  $\underline{E}$  and actual values. Thereafter the errors appear to cycle around an average value that has converged to a value close to zero. This behaviour is entirely consistent with the fifth law, Conservation of Organisational Stability. It is also consistent with the eighth law and the feedback hypothesis. The outliers are, therefore, not to be seen as exceptions. They represent the preliminary release process during which the system dynamic parameters are being established. Figures 4 and 5 will show just how quickly and how firmly the dynamics is established. Beyond that point the dynamics takes over. Human action as, for example, in determining of the content of each release, has only a second order effect on the dynamics of growth. An improved model for future FS release planning can, therefore, be obtained by ignoring the build up period and computing  $\underline{E}$  from data beginning with rsn 6 and 7.

Turski pursued a different line of investigation to investigate the strength of the dynamics [tur96]. In the first instance he had computed  $\underline{E}$  from all twenty pairs of data points  $i = 1$  to  $i = 21$  as indicated by [2]. He then asked the question "How fast is the dynamics established? How many data pairs are required before  $\underline{E}$  has settled to a reasonably constant value?" Equivalently, from how many data point pairs starting with rsn 1 must  $\underline{E}$  be computed to achieve a good estimate for  $\underline{E}$  and an appropriately low standard deviation of the inverse square size prediction? The answer to these questions is illustrated by figure 4. This shows the average error in size prediction and its standard deviation derived from estimating  $\underline{E}$  by using successive subsets of  $j$  data point  $2 \leq j \leq 20$ . The plot confirms the conclusions reached in the comments on figure 3 and indicates the strength of the system dynamics effect. The dynamic parameter is firmly established after only some four or five releases. This is a most remarkable result. More statistical evaluation and similar data from other systems is, however, required before the conclusions can be generalised with full confidence.

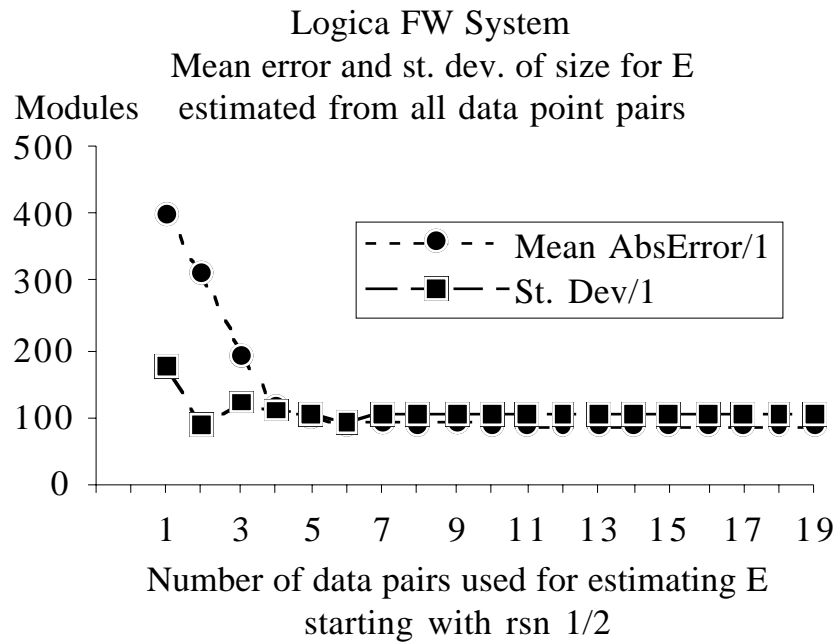


Figure 4. Mean error and standard deviation in size prediction for  $\underline{E}$  based on full data set

Repeating the experiment with the five initial outliers excluded as in figure 5 strengthens the conclusions, confirming the strength of the dynamics.

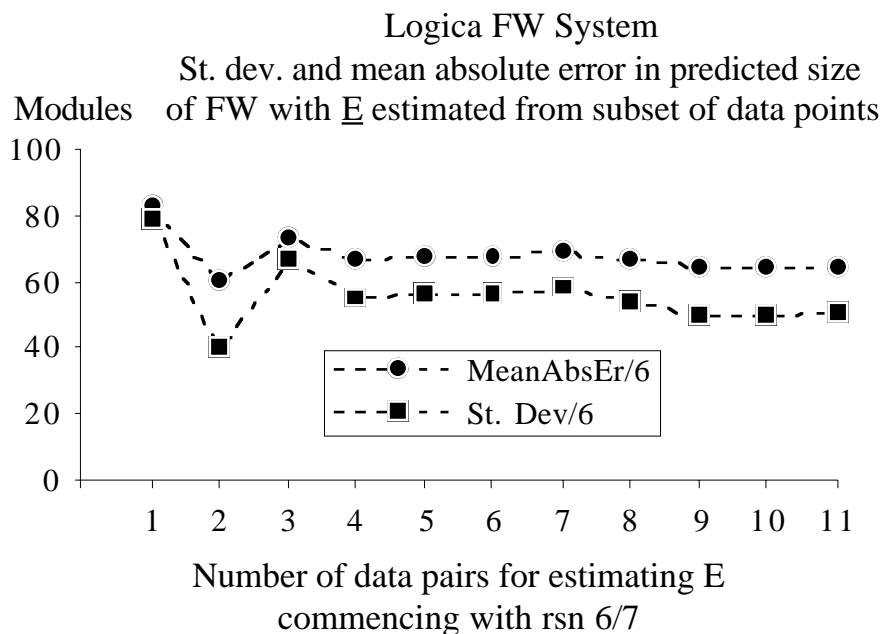


Figure 5. Mean error and standard deviation in size prediction for  $\underline{E}$  on data set that excludes the 5 initial outliers - note difference in scale relative to figure 4

The plots confirm the remarkable precision of the inverse square model and the constancy of "effort", as per the fourth law. In the absence of an alternative explanation, a standard deviation and average absolute error (ignoring the ripple) which is almost constant at under 80 modules for a system growing from 1800 to 2300 modules indicates the strength of the dynamics. Additional work is required to superimpose a fit to the ripple on the inverse square fit to produce an even better model. The FEAST/1 investigation will address these matters and seek general solutions.

Finally the absolute and average growth per release must be considered. Figure 6, indicates that the former also displays regularity. After the initial releases, the average appears well defined as

predicted by the fifth law, Conservation of Familiarity. The observed behaviour could, however, also be due to conservative planning by the Logica management and/or relatively constant demand by sales personnel and clients. No firm conclusion can therefore be reached as to the support FW data provides for the fifth law until an attempt is made in the future to implement a release that significantly exceeds the average growth. If and when this occurs and the consequences for users and on subsequent system releases is observed, resolution should be possible.

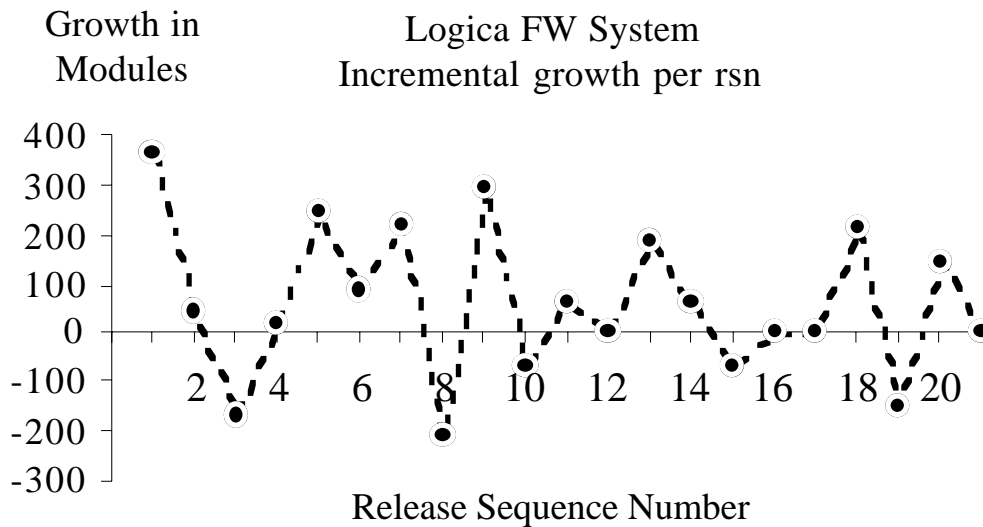


Figure 6. Incremental Growth

### 9 Summary

In summary, the FW data presently available and the minimal analysis so far undertaken does not yet provide a conclusive demonstration of anything. Alternative explanations of the problems being encountered in process improvement are discussed elsewhere [leh96c]. But the case for feedback is building. A more decisive judgment must await detailed exploration of more data on more systems. Meanwhile consideration of this first set of data has provided impressive support for the laws as indicated in table 2.

No.	Support	Indicator
I	?	More investigation is required to determine whether the growth trend of fig. 2 includes adaptation and change and so supports I or only functional growth as in VI
II	√	The inverse square law of growth, eq. 1, constant effort parameter $\underline{E}$ , figs. 2 and 3
III	√	Figures 2 and 3 but no tests for normality applied yet
IV	√	Ability to obtain close fit with constant $\underline{E}$ in equation (1)
V	?	Figure 6 which is similar to equivalent data in the OS/360 data. Conclusion is, however, tentative, awaiting observation of the consequences of a release whose content exceeds the average by a significant amount
VI	√	Demonstrated by the growth of as in figure 2 and analysis of its content
VII		No evidence for or against
VIII	√	Self stabilisation as in figs. 2 and 3 and as implied by figs. 4 and 5. Inverse square law, eq. 1. Proof that process is a feedback system requires identification of feedback mechanisms and cannot be inferred from quantitative data.

Table 2 Support for the Laws of Software Evolution from the FW Data

The data suggests that the dynamics of FW's evolution process is strong with the growth parameter, at least, fixed early in the life of the evolving system. If, as suggested by the phenomenological analysis, the results from OS/360 and FW can be generalised, monitoring the initial releases of a system will generate data from which planning tools can be derived to identify attainable release parameters and so facilitate significant improvement of the release planning and management process. This, itself, represents significant improvement. More is needed and is believed possible.

The potential may be illustrated by reference to resource and cost estimation tools such as Cocomo [boe81,95]. These tools are essentially pragmatic. They estimate future resource requirements for a defined project on the basis of selected parameters and past experience, not on the objective needs of each project. The FEAST hypothesis and the process dynamics it implies fully explains their success, why they work; why indeed it is difficult to do better. Once the dynamic parameters of an organisation, a project or a process have been established, they determine resource expenditure and global productivity. Cocomo works because of, not despite, the organisational dynamics. The FEAST hypothesis is therefore a full conceptual justification for the legitimacy of the Cocomo approach *in the absence of full knowledge and understanding* of the process dynamics. A model of the dynamics of the organisation and project would provide an objective basis for schedule and cost estimation and prediction. In its absence organisational and project history provide a subjective basis. Their use for objective systematic improvement is more limited. In the presence of strong dynamics, objective assessment of resource needs or productivity rates has little meaning unless based on the dynamic characteristics. Objective tools can emerge only when the dynamic characteristics of an organisation and its processes are fully mastered. Only then can one hope to change the underlying dynamics and so reveal the potential and means for process improvement.

The results reported here are persuasive but not conclusive. To broaden the FW analysis requires more data. Its generalisation requires similar data from other systems, different organisations and different developments. Moreover the black box approach described here must, and will in FEAST/1, be complemented by a white box, or systems dynamics approach. This will permit modelling of the global software process structure, the isolation of feedback mechanisms, determination of their characteristics and identification of constraints they impose on process improvement. Demonstration of the extent of the feedback phenomenon and of the constraints it imposes on improvement will trigger and facilitate the development of means for its mastery and exploitation. Success in this regard will have profound implications on system evolution, release planning and project and process management. Mastery of the software process system dynamics will represent a major advance in the development of a theoretical framework for software process technology, in advancing that technology and in advancing the capability for radical process improvement.

## 10 Acknowledgments

My grateful thanks are due to Dewayne Perry, Vic Stenning and Wlad Turski who put up over many days of heated discussion with my obstinate refusal to abandon my beliefs. The responsibility for all the assertions made here is mine. But it was the continuing close co-operation with them for a number of years that has made this paper, and, more importantly, the results, possible. Also to Wlad for permitting me to quote from his work before publication of his paper. Thanks also to Logica plc for providing access to their FW data and for permitting its publication in the present form. Finally to EPSRC for their grants to support FEAST/1 and the continued association of Perry and Turski with that endeavour. Also to the UK DTI for an earlier grant under the ESF Extension supporting the initial FEAST investigation.

## 11 References<sup>5</sup>

- [abd91] Abdel-Hamid T and Madnick SE, *Software Project Dynamics - An Integrated Approach*, Prentice Hall, Englewood Cliffs, NJ 07632, 263 p.
- [bec94] Becker RS, Hall B, and Rustem B, *Robust Optimal Control of Stochastic Nonlinear Economic Systems*, J. of Economic Dynamics and Control, n. 18, 1994, pp. 125 - 148
- [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
- [ben56] Bennington HD, *Production of Large Computer Programs*, Proc. Symp. on Advanced Computer Programs for Digital Computers, sponsored by ONR, June 1956, Republished in [boe81] Boehm B W, *Software Engineering Economics*, Prentice Hall, Englewood Cliffs, NJ, 1981
- [boe76] Boehm BW, *Software Engineering*, IEEE Trans. on Comp., v. C-5, n. 12, Dec. 1976, pp. 1226-1241

<sup>5</sup> Papers identified by a \* in the references listing are reprinted in [leh85].

- [boe95] Boehm BW, Clark B, Horowitz E, Westland C, Madachy R, and Selby R, *Cost Models for Future Software Life Cycle Processes: Cocomo 2.0*, to appear in Annals of Software Engineering Special Volume on Software Process and Product Measurement, J D Arthur and S M Henry (eds), J C Balzer AG, Science Publishers, Amsterdam, 1995
- [boe76] Boehm BW, Software Engineering, IEEE Trans. on Comp., v. C-5, n. 12, Dec. 1976, pp. 1226-1241
- [boo86] Booch G, *Object-Oriented Development*, IEEE Trans. on Softw. Engineering v. 12, n. 12, pp. 211 - 221
- [bux70] Buxton JN and Randell B (ed), *Software Engineering Techniques*, Report on a Conference, Sponsored by NATO Sc. Comm., Rome, 1969, Sc. Aff. Div., NATO, Brussels 39, 1970, 170 p.
- [cho81] Chong Hok Yuen CKS, *Phenomenology of Program Maintenance and Evolution*, PhD Thesis, Dept. of Comp., Imp. Col., 1981
- [dij68] Dijkstra EW, *GOTO Statement Considered Harmful*, Letter to the Editor, CACM, v. 11, n. 11, Nov. 1968, pp. 147 - 8
- [fea94,5] *Preprints of the FEAST Workshops*, MM Lehman (ed.), Dept. of Comp., ICSTM, 1994/5
- [gra96] Gray EM and Smith WL, *On the limitations of Software Process Assessment*, CS Dept., Glasgow Caledonian Univ. Glasgow G4 0BA
- [[gri78] Gries D, *Programming Methodology - A Collection of Articles by Members of IFIP WG2.3*, Springer Verlag, New York, 1978, 437 p.
- [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*, ICST DoC Res. Rep. 83/13, 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*, Academic Press, London, 1985, pp. 538
- [leh87] Lehman MM, *Process Models, Process Programs, Programming Support - Invited Response To A Keynote Address By Lee Osterweil*, Proc. 9th Int. Conf. on Softw. Eng., Monterey, CA, 30 March 2 Apr. 1987, IEEE Comp. Soc. pub. n. 767, IEEE Cat. n. 87CH2432-3, pp. 14 - 16
- [leh91] *id.*, *Software Engineering, the Software Process and Their Support*, IEE Softw. Eng. J. Spec. Iss. on Software Environments and Factories, Sept. 1991, v. 6, n. 5, pp. 243 - 258
- [leh95] *id.*, *Process Improvement - the Way Forward*, Proc. CAiSE 95, Jyvaskyla, June 1995, Lect. Notes in Comp. Sci., Springer Verlag, pp. 1 - 11 (an early precursor of this paper)
- [leh96a] 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.
- [leh96b] Lehman MM and Stenning V, *FEAST/1: Case for Support*, ICSTM, March 1996
- [leh96c] Lehman MM, *Laws of Software Evolution Revisited*, Proc. EWSPT'96, Nancy, 9 - 11 Oct. 1996
- [leh96] Lehman MM, *Process Modelling - Where to?*, "Most Influential Paper of ICSE 9" Award, Proc. ICSE 19, Boston 18 - 20 May 1997
- [maj93] Major J, *Keynote Address*, ICSE15, Baltimore, 17 - 21 May 1993
- [nau69] Naur P and Randell B, *Software Engineering - Report on a Conference*, Sponsored by the

NATO Science Committee, Garmisch, 1968, Scientific Affairs Div., NATO, Brussels 39, 1969, 231 p.

- [ost86] Osterweil L, *Software Processes are Software Too*, Iteration in the Software Process, Proc. of the 3rd Int. Proc. Workshop, Breckenridge, CO, 17 - 19 Nov. 1986, IEEE cat. n. TH0184-2, IEEE Comp. Soc. order n. 709, 1987, pp. 79 - 80
- [ost87] *id.*, *Software Processes are Software Too*, Proc. 9th Int. Conf. on Softw. Eng., Monterey, CA, 30 March - 2 Apr. 1987, IEEE Comp. Soc. Pub. n. 767, IEEE Cat. n. 87CH2432-3, pp. 2 - 13
- [ost97] *id.* P, "Most Influential Paper of ICSE 9" Award, Proc. ICSE 19, Boston 18 - 20 May 1997
- [pau93] Paulk MC, Curtis B and Chrissis MB, *Capability Maturity Model for Software, Version 1.1*, Software Engineering Technical Report, CMU/SEI-93-TR, Feb. 24 1993
- [spw84] Potts C (ed), *Proceeding of the Software Process Workshop*, Egham, Surrey, UK, Feb. 1984. IEEE, cat. n. 84CH2044-6, Comp. Soc., Washington D.C., ord. no. 587, pp. 27 -35
- [roy70] Royce WW, Managing the Development of Large Software Systems, *IEEE Wescon*, Aug. 1970, 1 - 9
- [sen90] Senge PM, *The Fifth Discipline*, Currency Doubleday, New York, NY, 1990, 423 p.
- [tur96] WM Turski, *Reference Model for Smooth Growth of Software Systems*, IEEE Trans. on Soft. Eng. v.22, n. 8, August 1966
- [wil51] Wilkes MV, Wheeler DJ and Gill S, *The Preparation of Programs for an Electronic Digital Computer*, Addison Wesley Press Inc., 1951, 167 pp.
- [wir71] Wirth N, *Program Development by Stepwise Refinement*, CACM, v.14, n.4, Apr. 1971, pp.221-227
- [you79] Yourdon E and Constantine LL, *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*, Prentice Hall, Englewood Cliffs, NJ, 1979