ELSEVIER

Information and Software Technology 38 (1996) 681–686

INFORMATION AND SOFTWARE TECHNOLOGY

Feedback in the software evolution process

M.M. Lehman*

Department of Computing, Imperial College of Science, Technology and Medicine, Prince Consort Road, London SW7 2BZ, UK

Abstract

Despite major advances in programming methodology, industry still faces major problems in developing and maintaining *E*-type software — software that implements computer applications in the real world. The paper suggests that a possible constraint on software process improvement arises from the fact that the global software process that includes technical, business, marketing, user and other activities constitutes a multi-loop, multi-level feedback system. To change the characteristics of such a system requires one to consider, design or adapt and tune both forward and feedback paths to achieve the desired changes in externally visible behaviour. It should, therefore, not come as a surprise that the overall improvements achieved fall far below expectations. After all, current world-wide process models and improvement activities focus primarily on the forward technical path and overlook the many feedback paths and the constraints that they impose on improvement of the project. A recently launched project, FEAST, based on a hypothesis by the same name, will investigate these observations and their consequences on the basis of an international collaborative investigation.

Keywords: Software process; Process improvement; Feedback systems; FEAST

1. Introduction

The means used to develop *E*-type[1] software have been significantly advanced and enriched over the last thirty years. Continuing efforts to improve the software development process have produced new concepts, methods, techniques and tools. High level languages, structured programming, abstract data types, formal methods, non-procedural programming paradigms, object orientation, CASE and support environments exemplify innovations which, each in their turn, were expected to overcome problems that have frustrated the consistent and cost effective on-time industrial development of functionally satisfactory and reliable software. Introduction of such innovations have certainly yielded major benefit. The use of structural programming and high level languages, for example, has greatly improved the program design process. The work in formalization has driven and directed programming methodology yielding major advances in Computer Science and in the design of computational processes. It has also had a direct impact on the programming process and, in particular, has provided a solid base for its mechanization. More generally, individual and small group effectiveness in executing specific development tasks have advanced considerably.

2. Process improvement

The introduction of improved technology has enabled major growth in the size and complexity of computing systems. But none of the innovations has proven to be a panacea, the silver bullet [2,3]. Their introduction into industrial practice has not yielded an industrial capability for consistent, planned development of quality software, has not resulted in major productivity growth or cost reduction nor in faster response to user needs. It has proven equally difficult to achieve major improvement in processes used to maintain systems satisfactory as user needs, desires and expectations change and as operational domains evolve. Software *evolution*, whether from application concept to first implementation or from release to release still relies on industrial processes that are far from satisfactory.

It may, of course, be unreasonable to expect order of magnitude improvement, however defined, from individual innovations or in the software process as a whole though major industrial software developers

* Tel: +44 (0)171 594 8214. Fax: +44 (0)171 594 8215. Email: mml@doc.ic.ac.uk

[1] *E*-type software is informally defined as software that implements an application or addresses a problem in the real world [1].

PII S0950-5849(96)01121-4

believe otherwise [4]. The fact that benefits are anticipated is no reason why they should be obtained. Software development is, intrinsically, a creative intellectual activity calling for significant human involvement, judgement and decision taking. It can be supported but not replaced by mechanization and this, in itself, limits potential global quality, productivity and responsiveness improvement. Moreover, such a process cannot be fault free. The occurrence of faults, of whatever type and however caused, are a major impediment to the achievement of increased productivity or responsiveness though use of technologies such as ever higher level languages, application specific languages and reuse may reduce their rate of occurrence. The fact remains that more sophisticated industrial and commercial organizations still regard software development as a hazardous enterprise and a far from perfect technology.

It is not difficult to explain why adopting individual innovations has not led to major progress in consistently achieving reliable, responsive and cost effective evolution of quality *E*-type software despite major R&D investment and the success of such innovations when applied to *S*-type[2] program development. The generality of the lack of success, however, suggests that a global constraint may lie at the root of the problem. Identifying one or more common causes would, of course, constitute important progress in the search for process improvement.

3. Constraints on improvement

One such cause is immediately apparent. The activity devoted to individual innovations such as those mentioned in the opening paragraph, represents only a small fraction of all the action embodied in the process. It cannot, therefore, be expected to have major impact at the global level. Activity at that level includes the many steps of technical development but, equally, application analysis, process and project management, marketing, usage, user support, enterprise management and so on. All these, and many more, have an impact and bearing on the characteristics and outcome of the *process* and its *product*. On the other hand, the common cause may stem from the size and complexity of the organisations needed to define, develop, market and use the systems, or from the number and diversity of people involved in *E*-type evolution. Does the difficulty of achieving significant process improvement spring from the complexity or other characteristics of *E*-type applications and systems? Are there, indeed, other characteristics of *E*-type software and of the associated evolution process to be identified and taken into account when designing or improving the process and the organizations that implement it? This paper is intended to draw attention to one further factor that may well play a major role in constraining process improvement directly and through its impact on the other potential explanations offered above.

4. *S*-type and *E*-type software

An initial clue emerges from the contrast between the significant advances that have been achieved in *S*-type programming and the continuing problems haunting industrial software development and maintenance, the evolution of *E*-type software. The *S*-type approach is well represented by the work of the IFIP WG 2.3 group on programming methodology [5]. This recognizes the need for iteration and backtracking over individual process steps, or even between a number of steps, to rectify errors or escape from *blind alleys*. Feedback from the output of the development or change process to the specification is, however, meaningless since, by definition, the latter is fixed. *Correctness* of the final program relative to that specification is the *only* valid criterion of success in implementation. If or when for any reason the completed program, while technically correct, does not meet client needs a *new* specification, possibly derived from the previous one, must be generated and a *new* program to satisfy it must be developed. This program too may be derived from the earlier program. But technically it is new since it is defined by a new specification. Anticipating the hypothesis presented below, one may observe that feedback from the *S*-type process output to its input has a long delay and low gain. Any evolution dynamics is broken up, staticized, by the decreed inviability of the specification.

In the context of *E*-type programs, on the other hand, *correctness* is meaningless. The real world application and its operational domain are potentially unbounded. Their descriptions cannot be absolutely or completely formalized to permit a correctness demonstration. Instead the criterion of acceptability is *user satisfaction* with each program execution [1,6]. The limits of the application and the operational domain are dictated by pragmatic considerations, resources and technology for example, and are ultimately defined by the implemented program. Feedback expressing individual or group dissatisfaction of developers, users, marketeers or executives for example, will impact future development of the program. The very nature [6,7] of real world applications and of the *E*-type software that models and implements them sets up continuing pressure for change and system evolution. That is, development and usage changes perceptions, leads to fresh insight and understanding, generates new needs and opportunities, changes expectations and criteria of satisfaction. The

[2] *S*-type software is required to be *correct* in the full mathematical sense with respect to a fixed specification [1].

evolution process is a *learning process*, driven and controlled by information gathered, interpreted and used to direct subsequent activity or to modify results of previous activity [8,9].

One aspect of that process was first studied in the early 1970s and led to the identification of an evolution dynamics [10] and to the formulation of laws relating that dynamics to human and organizational behaviour [11,12]. These Laws of Software Evolution, and additional ones since formulated, were derived by interpretation of models derived from measurement of the dynamics. They reflect human and organizational behaviour and characteristics which explains why they must be regarded as laws by the software engineering community. More recently organisational and managerial aspects of software evolution dynamics have been studied and methods for their exploitation developed [13].

5. Dynamics in software evolution

The results of these various studies provide the basis for an emerging theory of software evolution [6,9,13]. The theory is based on measured data stemming from several systems of different origins [1] and is supported by other factual evidence. The earliest measurements related to the growth of the IBM OS/360 operating system from birth to its fission into VS1 and VS2. The example of that data shown in Fig. 1 [1,9] provides the earliest phenomenological evidence of the regular dynamic nature of software evolution and its feedback-system-like characteristics.

The cyclic pattern discernible in Fig. 1 is characteristic of self-stabilizing feedback systems. As observed at the time [19] "...: the plot ripple is typical of a self-stabilizing process with positive and negative feedback loops. From a long-range 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 ...". The period of instability beyond release twenty representing OS/360 fission into VS1 and VS2 some months after the 1972 paper was published is equally indicative of the feedback nature of the software release process. The oscillatory behaviour indicates a loss of control over system evolution. It is consistent with all known facts that this chaotic behaviour was triggered by over ambitious growth targets, stemming from excessive positive feedback.

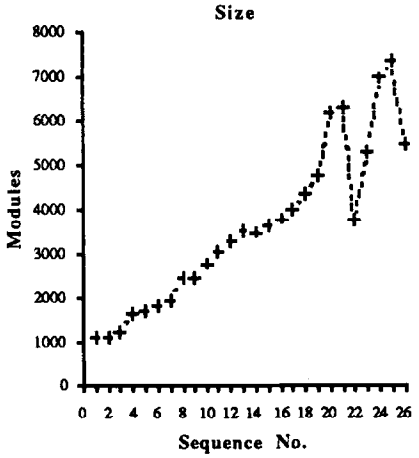

Fig. 1. The growth of IBM's OS/360 as a function of release sequence number.

The 1969–1972 observations from which the feedback nature of the software process was first inferred were restricted to evolution at the release level. Observation may, however, be extended to other levels of the software process. Learning and feedback play a major role in determining many, if not all, of the dynamic aspects of *E*-type software evolution at all levels. The *E*-type evolution *process* constitutes a multi-loop system, with both positive and negative feedback. The characteristics of the feedback loops and mechanisms determine process dynamics. The process may therefore be expected to display the stable behaviour that is the hallmark of feedback systems in general. Externally observable system properties and behaviour remain relatively constant within specified limits over the operational range until instability sets in despite changes in the characteristics of forward path elements, the process environment and the operational environment.

6. The global process

These observations may have been interpreted as restricted to the technical transformation process that is followed to refine a computer application concept into a solution system. After all the 1970s investigation concentrated on technical development. Their relevance is, however, much wider. Business management, marketing, customer support, project and process management all apply feedback as, for example, in the use of monitoring and reporting mechanisms as checks and balances. Similarly, organizational processes use feedback procedure to ensure, for example, steady business and organizational growth with disciplined product evolution as user experience and changing client needs are fed back to development organizations.

Similar mechanisms moderate evolution of the industrial software process. In general therefore, changes to forward path elements of the software process cannot be expected to produce major global improvement unless accompanied by commensurate changes to related feedback mechanisms. Software process improvement must, therefore, be pursued in the context of the total process

domain; business management, budget control, marketing, customer support, personnel, project, process and information management, technical development, process improvement and monitoring of all these. Such a broad focus, treating the process as a feedback system provides a realistic framework already exploited [13] for the study of process dynamics. As already observed, the dynamics is also implicit in the laws of software evolution [9,11,12] which, while inferred from technical data, were interpreted in terms of human and organisational behaviour and, in particular, in the organisational information feedback that drive and control the evolution process.

7. The FEAST hypothesis

These observations provide the *common cause* suggested above. The innovations listed in the opening paragraph were all forward path mechanisms. Yet their introduction into practice did not, in general, include a comprehensive feedback review process. Thus though that introduction may have changed local process properties, given the feedback stability property, it should not come as a surprise that the wider impact was far less than expected. This observation has been formalized in a hypothesis, FEAST, which implies that the common factor inhibiting major software process improvement is undue concentration on forward path innovation. In a feedback system, global changes cannot, in general, be achieved without explicit attention to feedback characteristics.

However, feedback, even on its own, has the potential to produce significant benefit. This is confirmed for the software process where innovative feedback based techniques such as *inspection*, *reviews*, *prototyping*, *incremental* and *evolutionary development* and *measurement* have, even in limited use, shown enough promise to constitute positive evidence for the hypothesis.

Note that the hypothesis consists of two separate and distinct assertions with the second arising if and only if the first is valid.

Assertion 1. The software evolution process for *E*-type systems, which includes both software development and its maintenance, constitutes a complex feedback learning system.

Comment. This assertion is undeniable. Hence, to obtain full benefit from a forward path change to the software process will generally require feedback paths and mechanisms to be added, removed or modified. This is well known. The question to be asked and answered is whether this process can be systematized; whether software process feedback design can be disciplined. The problem arises because of the human role in the process. Humans observe, interpret, verbalise, transform, communicate, assess, decide, control and apply the forward and feedback process information. They manipulate the information fed over the feedback paths that link the activities, organisations, spaces [14] and people involved in and responsible for the evolution process. Humans participate in and observe the software in execution, use the results of that execution and feed back their observations and experience to the development organisation. Is their involvement so ingrained, so extensive, so individual, so judgemental and so creative that meaningful and exploitable formalization and modelling with optimized design and integration of the feedback mechanisms is beyond reach, at least for the moment? Concern is not with the validity of the assertion but with its relevance, its significance and with development of means for its exploitation.

A crucial tool in providing means for its exploitation is the ability to model the process. Modelling techniques employed in current software process modelling activity do not, generally, provide the necessary facilities since they have not sought to reflect detailed feedback properties. But relevant formalisms and methods have been developed in other areas. Comprehensive formal theory and methods for automatic feedback systems design and control for both continuous and stochastic systems is embodied in, for example, control and dynamic systems theories. In the past thirty years or so both have been extended and more or less successfully applied to systems involving humans; economic systems and organizational dynamics for example. There have also been some studies of the application of control theory to software development [9,13,15]. There exists, therefore, a prima facie case indicating that despite intimate human involvement formal models fully reflecting feedback mechanisms may be successfully developed and applied to the design and improvement of software processes. The assertion implies that such models are essential for major process improvement; that one must seek a representation or representations that permit integration of process models, as required, of *all* aspects of the process.

The process may be observed, described modelled, analysed and changed in many spaces and at many levels of detail. Feedback occurs when information originating at the output of some process element is injected after some delay to the input of that or an earlier element. At lower process levels where individual action in the software process is relatively isolated feedback refers to the transfer of a single, in some sense spontaneous and unpredictable, package of information for use as seen fit by the recipient. Where, as at higher levels of the process, a continuing stream of (discrete) information is involved, *feedback* in the full engineering sense of the term, is established. The two usages do not really differ. They simply assume different situations. Rigorous representation of the former case is difficult and prediction of its consequences may be impossible. The analysability of the latter despite the non-determinacy of human involvement, stems from the fact that the total information flow

in the process system generally involves many decisions and many decisors. The information fed back is a composite of many inputs which, whilst not totally predictable or independent, are amenable to meaningful statistical representation and analysis. The consequent system behaviour has been shown to display statistically *normal* properties [16]. It is therefore reasonable to expect that at these levels of the software process, control theoretic and statistical process models reflecting system dynamics can be developed for use on their own or in conjunction with modelling techniques currently in vogue. When statistical representation is not possible new formalisms permitting representation of the feedback mechanisms will have to supplement current process modelling techniques. At this level simulation techniques would likely constitute an important element of the design and evaluation process.

Assertion 2. The feedback nature of the evolution process *explains*, at least in part, the failure of forward path innovations such as those introduced over the last decades to produce impact at the global process level of the order of magnitude anticipated.

Comment. This assertion does not deny that many, if not all, of the innovations yielded significant benefit at the local level, improving the effectiveness of individual process steps or activities significantly. High level languages, for example, have certainly increased the quality, productivity and predictability of program code development and of code changes by an order of magnitude. The assertion merely states that because of the constraining effect of feedback mechanisms such benefit did not extend as fully as expected to the global level. This general failure need not, of course, be due to this common cause. Instead it may be due to reasons specific to each innovation. In view of the number of such failures a common cause must, however, be suspected. The feedback hypothesis provides an explanation that is consistent with an established property of other feedback systems. It is, therefore, of interest to examine the innovations individually in the context of processes within which they have been employed to determine whether limitations to global improvement achieved can be attributed to the feedback nature of the software evolution process or can be overcome by modifying or providing additional feedback mechanisms. Failure to succeed in either would not prove the hypothesis invalid. It would cast doubt on its practical relevance.

8. Summary

In summary, from the fact that feedback systems display a degree of global stability and resistance to change dependent on the detailed characteristics of their feedback mechanisms and that the software evolution process constitutes a feedback system one must assume that the benefits obtained from (the observable consequences of) innovative changes to forward path methods techniques or tools in the software evolution process will be limited (may be increased by) feedback effects. The extent and degree of the constraining effect will depend on the characteristics of the many individual feedback paths and on the interactions between them. It may equally be anticipated that the global benefit obtained from improvements in forward path technology can, generally, be increased by attention to (adjustment of) the characteristics of appropriate feedback paths or mechanisms. It is tempting to suggest that the feedback phenomenon explains the relatively slow progress made in global software process improvement despite the many innovative concepts that have been introduced into the forward path technology; that this is the "common cause" hypothesized in the opening paragraphs of this paper. Whether this is indeed so remains to be demonstrated as does the question whether, if true, it can be systematically exploited. These, and related, questions are now being explored with International collaboration, in a project, FEAST, supported in its initial stages by a grant from the UK Department of Trade and Industry. If successful, the project may be expected to have a profound impact on the software development and maintenance (software evolution) processes and on the process of process improvement.

9. The FEAST project — (Feedback, Evolution And Software Technology)

This recently launched project is exploring the feedback hypothesis to verify it and to search for ways in which it may be exploited. As observed above, the basic fact that the software evolution process constitutes a learning based feedback system is self-evident. What must be investigated and demonstrated is the extent to which feedback phenomena have constrained the benefit derived from the introduction of innovative concepts, methods, tools and techniques in forward path mechanisms. Ideally one should be able to identify specific feedback paths that inhibited or damped the benefit obtained from individual innovations and to identify changes to the feedback structure and mechanisms that would produce additional benefit. At the same time it could be most rewarding to exploit the insight expressed in the hypothesis; to develop methods, techniques and tools whereby the process, including its feedback mechanisms, may be modelled, evaluated and implemented or changed to exploit the potential benefits of each innovation to the fullest extent.

As a first step it is intended to model the process and its properties using appropriate techniques and representations to expose the role and impact of feedback in software evolution. Models will be derived both from

process theory and from observation, measurement and analysis of industrial processes. Detailed examination of the role and contribution of people in such mechanisms will be included. A necessary precursor to this modelling activity is the adoption of formalism that permits adequate representation, at various levels of detail, of software processes with their feedback loops and mechanisms. Exploration of suitable techniques and representations must include control theoretic and system dynamics approaches as well as the formal languages such as those currently used in process modelling. Nevertheless, the proposed modelling activity will differ radically from that currently in vogue. The latter tends to divert attention, even hide, global process activities and properties. Project FEAST will focus on them.

The insight and understanding developed in the early stages of this integrated analysis of current software process technology will lead to process evaluation and improvement in terms of both forward and loop properties. Exploitation of existing and emerging development and support technology must be enhanced so as to yield improved process attributes. Methods, techniques and tools for the design and evaluation of feedback control mechanisms must be developed. New and improved mechanisms exploiting the potential of feedback must be developed. Finally, lessons learned must be applied to the extension of process theory and the generation of principles and guidelines that will facilitate the transfer of results of the study to software engineers responsible for design, support and improvement of the software process, to software developers and to their managements in industry and elsewhere.

10. Postscript

The above paper is essentially the text as prepared for the CSR 11th Workshop on Software Evolution: Models and Metrics, Dublin, 7–9 September 1994. Since that time work on the FEAST hypothesis and its implications have continued and significant progress made [17–20]. An EPSRC grant to cover a two year FEAST/1 project has now been awarded and work under it with strong industrial collaboration will begin in the autumn of 1996.

References

[1] M.M. Lehman, Programs, life cycles and laws of software evolution, Proc. IEEE Special Issue on Software Engineering, 68 (9) (September 1980) 1060–1076.

[2] F.P. Brooks, No silver bullet — essence and accidents of software engineering, in Information Processing 86 — Proc. IFIP Congress, Dublin, 1–5 September 1986, Elsevier, London, pp. 1069–1076.

[3] W.M. Turski, And no philosophers stone either, Information Processing 86, Proc. IFIP Congr., Dublin, 1–5 September 1986, Elsevier, London, pp. 1077–1080.

[4] J. Major, Keynote Address, ICSE15, Baltimore, MD, 17–21 May 1993 (unpublished).

[5] D. Gries, Programming Methodology — A Collection of Articles by Members of IFIP WG2.3, Springer Verlag, New York, 1978.

[6] M.M. Lehman, Software engineering, the software process and their support, IEE Soft. Eng. J. Special Issue on Software Environments and Factories, 6 (5) (September 1991) 243–258.

[7] W.M. Turski, Specification as a theory with models in the computer world and in the real world, Infotech State of the Art Report, 9, No. 6, 1981, pp. 363–377.

[8] M.M. Lehman, The programming process, IBM Res. Rep. RC 2722, IBM Res. Centre, Yorktown Heights, NY, September 1969; also in [9], pp. 39–83.

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

[10] L.A. Belady and M.M. Lehman, An introduction to program growth dynamics, in W. Freiburger (ed.), Statistical Computer Performance Evaluation, Academic Press, New York, 1972, pp. 503–511.

[11] M.M. Lehman, Programs, cities, students — limits to growth, Imperial College Inaugural Lecture Series, vol. 9, 1970–1974; also in [12], pp. 42–49 and [9], pp. 133–163.

[12] M.M. Lehman, Laws of program evolution — rules and tools for programming management, Proc. Infotech State of the Art Conf. on Why Software Projects Fail, 9–11 April 1978, pp. 11/1–25.

[13] T. Abdel-Hamid and S.E. Madnick, Software Project Dynamics — An Integrated Approach, Prentice Hall, Englewood Cliffs, NJ.

[14] S. Benford and L. Fahlen, A spatial model of interaction in large virtual environments, in Michelis, Simona and Schmidt (eds.), Proc. 3rd European Conf. on Comp. Supported Cooperative Work — ECSCW '93, Milan, 1993, Kluwer Academic Publishers, pp. 109–124.

[15] C.M. Woodside, A mathematical model for the evolution of software, ICST CCD Res. Rep. 79/55, 1979; also in J. of Sys. and Soft., 1 (4) (October 1980) 337–345, and in [9], pp. 339–354.

[16] C.K.S. Chong Hok Yuen, Phenomenology of program maintenance and evolution, PhD Thesis, Dept. of Computing, Imperial College, London, 1981.

[17] M.M. Lehman, Evolution, feedback and software technology, Position Paper — SPW9, Proc. 9th Int. Soft. Process Workshop, 5–7 October 1994, IEEE Computer Society, 1995.

[18] M.M. Lehman, Feedback, evolution and software technology, Soft. Proc. Newsletter, IEEE Computer Society (Spring 1995), 3–6.

[19] M.M. Lehman, Process improvement — the way forward, Proc. CAiSE 95, Jyvaskyla, June 1995, Lect. Notes in Comp. Sci., Springer Verlag, pp. 1–11.

[20] M.M. Lehman, D.E. Perry and W.M. Turski, Why is it so hard to find feedback control in software processes? Invited Talk, Proc. 19th Australasian Comp. Science Conf., Melbourne, Australia, 31 January–2 February 1996, pp. 107–115.