# Adapting and Extending Empirical Studies to the Global Software Process

**M M Lehman** **J F Ramil** **G Kahen**
Workshop on "Beg, Borrow, or Steal: Using Multidisciplinary Approaches in Empirical Software Engineering Research"
ICSE 2000, Limerick, Ireland, June 5, 2000
Dept. of Computing
Imperial College of Science, Technology and Medicine
180 Queen's Gate, London SW7 2BZ
tel +44 (0) 20 7594 8214 fax +44 (0) 20 7594 8215
{mml, ramil, gk}@doc.ic.ac.uk http://www-dse.doc.ic.ac.uk/~mml/feast

## Abstract

Industrial global software processes include the activities of developers, users, sales and support personnel and their managers. Equally important is the feedback between them. Findings of empirical software engineering and other studies may not scale-up to such processes. Their study and that of the many environments in and for which they are pursued may be tackled by combining top-down and bottom-up approaches, broadening the scope and scale, and by following multidisciplinary investigative methods, tools and experience. Some of the latter may be inspired in numerical ecology, a discipline that suggests hints at the solution of issues inevitably emerging during broad scope and wide scale investigations.

Keywords: empirical studies, global software process, numerical ecology, software evolution

## 1 Introduction

In 1993 one of the authors (MML), considered why, despite continuing developments in software engineering such as structured methods, CASE, object orientation and so on, it is, in general, so difficult to achieve sustained process improvement. He realised that an answer has been initially suggested in a 1972 study [2] of the evolutionary behaviour of IBM OS/360. This study identified the software process as a self-stabilising feedback system. In such systems, *global* behaviour is significantly influenced by the interaction of feedback loops that, for example, constrain the externally observed effect of changes in forward paths. Since many process changes have local impact and affect forward paths only, one may expect that unless feedback mechanisms are changed appropriately, the externally observed impact of such techniques may be limited, externally unobservable or counterintuitive. The feedback observation was encapsulated in the FEAST (*F*eedback, *E*volution and *S*oftware *T*echnology) hypothesis, that in one of its formulations states: "As for other complex feedback systems, the dynamics of real world software development and evolution processes will possess a degree of autonomy and exhibit a degree of global stability". An alternative formulation, formalised as the eighth *law* of software evolution [4,8,10,11,17], is the following: "Software processes are multi-level, multi-loop, multi-agent feedback systems and must be treated as such to achieve sustained improvement for other than the most primitive processes" [13,14]. The hypothesis and its implications were extensively discussed at three international workshops [12]. Subsequently, two successive UK EPSRC funded projects, FEAST/1 [14] (1996-1998) and FEAST/2 (1999-2001) [15] were undertaken, the latter still in progress. A further workshop is planned for July 2000 [5]. The projects have been analysing data derived from a number of software systems being evolved in industrial processes. Interested readers can access summaries of the findings and publications to date through links at the project web site [4]. The present paper is mainly based on authors' accumulated experienced during these projects.

## 2 The Global Process

An *E*-type system, that is a system used in the real world, is judged by the results it delivers. Behaviour in execution, performance, functionality delivered, ease of use, reliability and so on are critical properties. Such system is evolutionary in nature, that is, it undergoes continuing fixing, adaptation and enhancement as long as it is in use. Requirements for such a system cannot be completely defined. In this it differs from an *S*-type system where the criterion of acceptability is that of correctness. By definition, it must be correct, in a mathematical sense, relative to an absolute specification [11]. Feedback is intrinsic to *E*-type processes and generally results in a sequence of versions or releases. Every release is built to a greater or lesser extent on code and documentation from the previous releases. For example, learning and experience derived from the development and use of previous releases is an important part of the input to the planning of each release. Apart from the most primitive, software evolution processes involve many stakeholders, clients, users, managers, sales personnel, development staff, and so on. Their aggregated activities and interactions *are* the *global* software process. Consider, for example, a process activity such as *application bounding* [11]. The views of the different stakeholders must be elicited, reconciled, merged, and limited to provide a release definition, whose implementation is technically feasible, within time, resource, budget, and other constraints. A process of successive blending and reconciliation of the many viewpoints provides a starting point for the development and subsequent evolution of the software system. The process changes the bounds of both the application and its domain until they represent a view of the proposed system (or system enhancement) that is acceptable to the decision-makers. The bounding process as above, and the release-based evolution process of which it is a part, have the structure of two nested feedback loops, as shown in figure 1, in which the output of a process is applied to and modifies its input. Feedback interactions can be found, however, at many other levels of the software process, including, for example, that of the teams and individuals that participate in the many roles involved in the global process. The immediate implication of the feedback observation is that software engineering must not only focus on forward path activities, but also consider feedback paths activities. This, of course, has implications in the context of empirical studies, which are briefly examined in the next section.

## 3 Empirical Studies of the Global Software Process

The issue introduced here is how the study of such global processes is approached and how the derived knowledge may be applied to progressively master issues such as process and product evolution management and realisation of definition, implementation and validation of the software. The multi-disciplinary nature of the study of the software process as a feedback system has been apparent since the impact of feedback on the software process was first recognised [2]. FEAST/1 and FEAST/2 focused on statistical and system dynamics perspectives [4]. The intrinsic properties of the problem demand a wider approach. Human issues, such as motivation, learning, experience, attitudes and assumptions, in management, development, marketing or end product usage must have a significant impact on the technical process and its product. Cognition and understanding and communication issues also play an important role. Organisational structure, its management and control, procedures, goal setting, achievement assessment, decision making, working methods and organisational culture all have significant influence [16]. Moreover, studies must consider the mixed environment in which the software process takes place. These, as depicted in figure 2, include human, business, economic, legislative, marketing, organisational systems and evolving technology, within which software engineering activities take place and with which they interact.

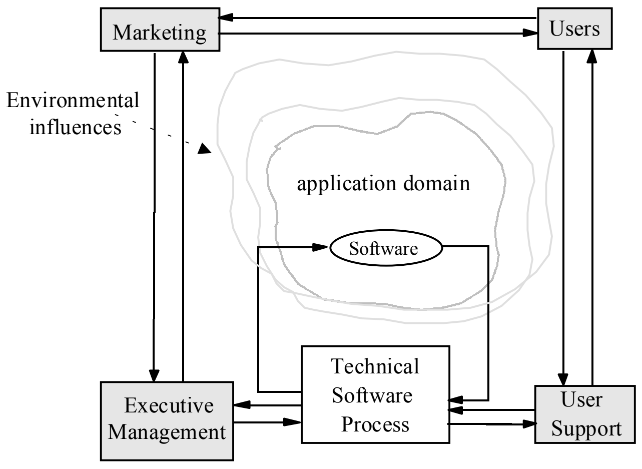

Fig. 1 - The Global Software Process as a Feedback System

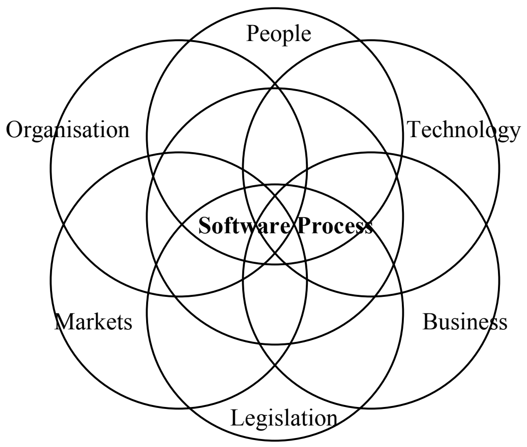

Fig. 2 - The Software Process and its Environmental Mix

It follows that one needs to adapt and extend the empirical studies to account for these wider influences. This is one of the major challenges faced by investigators. These, by-and-large, appear to have concentrated primarily on fine-grained process-local issues. In particular, such studies do not yet appear to have considered the environment within which software engineering takes place and with which it interacts. In fact, experimenters will acknowledge that the presence of a large number of context variables is one of the major difficulties in experimentation. Building knowledge through families of experiments has been suggested as a possible approach [1]. To solve this problem it has been proposed to replicate studies in different environments, vary context variables and capture and encapsulate commonalties across the environments [1]. This approach to building a body of knowledge is predominantly *bottom-up*. The issues that the global process raises suggest the need of pursuing *top-down* approaches [9]. The bottom-up approach appears to be most appropriate to the study of systems that are homogenous, or have simple, regular structure. Once individual elements have been identified, studied and understood to a certain degree, one may attempt to proceed to examine the properties and behaviour of more complex assemblies of these elements. In principle the approach could be applied successively to achieve understanding of ever larger and more complex systems. In practice, such studies may soon become ineffective, due, for example, to the unfeasibility of experimentation at a large scale, limitations of tools or mathematical intractability. There is, however, a top-down alternative. The latter studies the system or phenomenon of interest from the outside, following a top-down, outside-in approach. General trends and patterns are discovered and a set of attributes are identified, clarified, measured and modelled. This enables one to understand system's behaviour in the context of its sub-systems. This process eventually leads to an understanding of, and ability to control, the individual phenomena but in the context of their total environments. The global, systems, viewpoint has been applied with a degree of success, directly or indirectly, in many disciplines of the natural sciences, such as epidemiology, genetics, thermodynamics and information theory [9]. Our experience has shown that it is an essential approach to the empirical study of software processes [11]. In referring to bottom-up and top-down approaches here, we do not mean to imply that they are alternatives. In practice, both are required. It is really a question of their ordering and how they are interleaved.

The need to widen the focus of the studies and to follow a disciplined top-down approach is also illustrated by the following consideration. As Simon proposed in his *hierarchy theory* to explain the functioning of complex systems [20], he argued that stable complex systems are frequently hierarchical in structure. Such hierarchies are almost self-evident in some natural systems. In ecological systems, for example, this becomes apparent when one considers the environment as being primarily shaped by geo-morphological processes on land. Currents and winds determine fluid environments. These create localised spaces with similar conditions and interfaces to others. These broad scale mechanisms determine to a wide extent temporally and spatial patterns in biological systems. Within homogeneous zones, detailed structuring is shaped by predator-pray interactions, food availability, symbiosis and parasitism, for example [7]. Ecological systems are just one example of many other complex systems that reveal a hierarchy of mechanisms and influences. Subsequently, Simon extended this theory in his *sciences of the artificial* [21], of man-made systems, under which he included social systems. There are reasons to believe that global software processes display properties of hierarchical systems, with higher level influences may play a major role. The most evident is the one that emerges from the organisational decision making structures [23] in which the technical software process is embedded. Others may also exist. Hence, for a more complete and useful picture to emerge, empirical studies will need to address wider than process local issues [23].

Another challenge to future empirical studies must be also mentioned. So far software evolution studies such as [4,11] have concentrated on the study of similarities across systems and domains. This has led, for example, to the identification of the laws of software evolution [4,8,10,11,17] and to a discussion of the phenomenology of *E*-type processes. The interaction of software process feedback mechanisms may lead to patterns and structure, temporal and otherwise, in the processes and in the metrics therefrom derived. That is, the assumptions of classical statistics of independence of observations, made in many empirical studies, may not always hold [7]. For instance, one may expect that given appropriate environmental conditions, some software engineering methods may be more effective, in some sense, than others; some methods may flourish or succeed and others may vanish or fail. Thus, to further the investigation one must also look at the differences, at the heterogeneity and variability, and once identified, try to understand the mechanisms behind them.

## 4 Some Lessons from Numerical Ecology

In the context of the present workshop, that emphasises the application of tools from other disciplines, these authors wish to share some of their current perceptions in this regard. By seeking methods to address some of the FEAST/2 questions [15] they have identified that other disciplines such as *numerical ecology* contain methods, tools and experience potentially useful in the context of empirical software engineering. Quantitative models addressing software engineering issues have been investigated for some 30 years, as demonstrated by the long standing interest in cost modelling [18]. However, reported application of numerical methods in ecological and biological studies, that is, numerical ecology, started long before with a 1900 study [7]. Since then this discipline has, both, developed its own methods and applied methods from other disciplines. They include statistical and non-statistical methods (e.g. cluster analysis) and their combination. So far, several voluminous syntheses[1] have been published, for example, [7], that include pointers to relevant methods and guidelines to their use. Such syntheses represent a useful source of methods that can be applied to some aspects of empirical software engineering as, in particular, studies of software processes and software evolution [11]. Of particular interest are methods such as segmentation and chronological clustering. Some of these methods appear to fit within a top-down procedure for identification and encapsulation of patterns and structure in attributes of global software processes. One of the issues that would necessarily be associated with such application is the issue of *scale*. Results obtained in laboratory scale experiments investigating the effectiveness of certain software engineering techniques may not scale-up when these techniques are applied in an industrial process. In numerical ecology, for example, attention is been devoted to the issue of scale as exemplified by the following statement: "Scale is an important reference to help understand the difference between environmental management problems and the answers that may be found in ecological studies. Most studies are conducted at scales (extents) finer than those of natural or human related disturbances...Scaling up from studies to environmental problems is a challenge...New concepts and statistical tools must be developed..." [7, page 710],[22]. Similarly, due attention for such issues is needed in the context of empirical studies of software engineering.

In ecological spatial studies, such as the ones that search for patterns in the geographical distribution of unit objects, such as species or individuals, three elements have been identified as *descriptors* of the scale of a study:

- *grain size* - size of the elementary sampling units, such as resolution of an spatial study or of the smallest time unit considered in a time series study
- *interval* - distance between sampling units to cover a given geographical area or, in time series, the size of the time interval between observations
- *extent* - is the total size of the space covered by the study, or the total duration in time series.

Such descriptors must be defined in accordance with the size of the *unit objects,* and with the scope of action of the processes being investigated. The unit objects might be individual plants or animals, species or bacterial colonies. The processes could be those involved within the size of a patch of land, a lake or the area occupied by certain species and/over a given time period. In software engineering, definition of the descriptors and identification of the unit objects may not appear straightforward. Variables involved in software engineering studies, such as satisfaction, complexity, experience and motivation, are abstract. This imposes further challenges to their study. In spite of these, and possibly other, challenges, one can learn from field ecologists to carefully consider the scale of a study and to develop strategies to adapt the coverage of the study to wider and complex domains such as industrial global processes.

[1] At the present one just can hope that similar syntheses will in the future address empirical software engineering issues.

## 5 Final Remarks

Of course, when borrowing methods and tools from other disciplines one must exercise due care to understand the embedded assumptions and the conditions within which they fully apply. When we have mentioned numerical ecology, our intention has not been to suggest *a priori* similarities between the mechanisms at play in ecological systems and in the software process. Our emphasis has been on the methods, tools and experience that such discipline has evolved. One could however hypothesise that analogies may be fruitfully explored. For example, mathematical models based on differential equations such as the 'replicator equation' [19], which is able to reflect important ecological and other interactions [6], may also emerge in the software engineering domain. Surprising as it may appear, modelling the defect removal process as a prey-predator dynamic system has already been attempted [3].

Finally, one must accept that global processes involve human beings. These observe, think, learn, act and react and, hence, the study of systems that involve humans always will have to face an intrinsic degree of unpredictability [9]. Our knowledge about such processes, no matter what the discipline, approach, methods and tools must be always uncertain.

## References

1. Basili, V.R., Shull, F., and Lanubile, F., Building Knowledge through Families of Experiments, *IEEE Trans. on Softw. Engineering*, Vol. 25, No. 4, July/Aug. 1999, pp. 456 - 473
2. Belady, L.A. and Lehman, M.M., An Introduction to Program Growth Dynamics. In *Statistical Computer Performance Evaluation*, W. Freiburger (ed.), Academic Press, NY, 1972, pp. 503 - 511. Reprinted as chapter 6 in [11].
3. Calzolari, F., Tonella, P. and Antoniol, G., Dynamic Model for Maintenance and Testing Effort. In *Proc. of the International Conference on Software Maintenance ICSM 98*, Nov. 16 - 20, 1998, Bethesda, Maryland, pp. 104 - 112
4. *FEAST/2, Feeback, Evolution and Software Technology*, project web site, http://www-dse.doc.ic.ac.uk/~mml/feast
5. *FEAST 2000 International Workshop on Feedback and Evolution in Software and Business Processes*, London, U.K. July 10 - 12, 2000 http://www-dse.doc.ic.ac.uk/~mml/f2000
6. Lansing, J.S., Kremer, J.N. and Smuts, B.B., System-dependent Selection, Ecological Feedback and the Emergence of Functional Structure in Ecosystem, *J. Theor. Biol*., pp 377 - 391, 1998
7. Legendre, P. and Legendre, L., *Numerical Ecology*, 2nd. English Ed., Elsevier, Amsterdam, 1998
8. Lehman, M.M., *Programs, Cities, Students, Limits to Growth?*, Inaugural Lecture, in Imperial College of Science and Technology Inaugural Lecture Series, Vol. 9, 1970, 1974, pp. 211-229. Also in Gries, D., (ed.), *Programming Methodology*, Springer Verlag, 1978, pp. 42 - 62.
9. Lehman, M.M., Human Thought and Action as an Ingredient of System Behaviour. In Duncan, R. and Weston Smith, M. (eds.), *Encyclopedia of Ignorance*, Pergamon Press, Oxford, 1977. Also in [11] as chapter 11.
10. Lehman MM, Programs, Life Cycles and Laws of Software Evolution, *Proc. IEEE Special Issue on Software Engineering*, Vol. 68, No. 9, Sept. 1980, pp.1060 - 1076
11. Lehman, M.M. and Belady L.A., *Software Evolution - Processes of Software Change*, Academic Press, London, 1985
12 Lehman, M.M. (ed.) *Pre-prints of the (first) Three International FEAST Workshops*, available from links at [4].
13. Lehman M.M., Feedback in the Software Evolution Process, Keynote Address, CSR 11th Annual Workshop on Software Evolution: Models and Metrics. Dublin, 7-9th Sept. 1994, also in *Information and Software Technology*, sp. is. on Software Maintenance, vol. 38, no. 11, 1996, pp. 681 - 686
14. Lehman, M.M. and Stenning, V., *FEAST/1: Case for Support*, Part 2, DoC, Imperial College, Nov. 1995/March 1996, available from links at [4].
15. Lehman, M.M., *FEAST/2: Case for Support*, DoC, Part 2, Imperial College, August 1998, available from links at [4].
16. Lehman, M.M., *Mastering Process Feedback and Dynamics in Integrated Business, Technology and IT Evolution*, IRC Proposal, Part 2, DoC, Imperial College, June 1999, 7 pp. Available from links at http://www-dse.doc.ic.ac.uk/~mml/.
17. Lehman, M.M., Ramil, J.F., and Wernick, P.D., Metrics-Based Process Modeling with Illustrations from the FEAST/1 Project, to appear in Bustard, D., Kawalek, P. and Norris, M. (eds.) *Systems Modeling for Business Process Improvement*, Artech House, 2000
18. Nelson, E.A., *Management Handbook for the Estimation of Computer Programming Costs*, AD-A648750, System Development Corp., Oct. 31, 1966. Referred to by Boehm, B., in *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, N.J, 1981
19. Schuster, P. and Sigmund, K., Replicator Dynamics, *J. Theor. Biol*., Vol. 100, pp 533 - 538
20. Simon, H.A., The Architecture of Complexity, *Proc. Am. Philos. Soc*., vol. 106, 1962, pp. 467 - 482
21. Simon, H.A., The Sciences of the Artificial, 3rd. edition. The MIT Press, Cambridge, MA, 1996, 231 pp, first pub. in 1969
22. Thrush, S.F. et al, Scaling-up from Experiments to Complex Ecological Systems: Where to Next?, *J. Exp. Mar. Biol. Ecol*., Vol. 216, pp. 243 - 254
23. Witte, E. and Zimmermann, H.J. (eds.), *Empirical Research on Organizational Decision-Making*, North-Holland, 1986