Model-Based Assessment of Software Evolution Processes

G Kahen MM Lehman JF Ramil
Research Report
Dept. of Computing, Imperial College of Science, Technology and Medicine
London SW7 2BZ
tel +44 (0) 20 7594 8216 fax +44 (0) 20 7581 8024
{gk,mml,ramil}@doc.ic.ac.uk http://www-dse.doc.ic.ac.uk/~mml

Abstract

This paper argues that quantitative process models must be considered essential to support sustained improvement of *E*-type software evolution processes and summarises some of the experiences gained in the FEAST projects to date. Modelling guidelines are provided.

1. Introduction

Evolution is an intrinsic property of *E*-type[1] systems, that is, systems that support applications operating in the real world [leh85]. Evolution may occur as a series of versions or releases of one single system or as a sequence of versions of a family of systems or products. The process by which *E*-type software is evolved is generally as long-lived as the software itself, with sequential and incremental process changes becoming a reasonable approach to the improvement of such processes. In fact, every time the evolution process is re-enacted, as for example, every new release, there is an opportunity to introduce process changes and to assess their effects by means of comparison with process perfomance over past releases or process iterations. Some organisations have established infrastructures to capture and preserve process-related experiences derived from their software projects [cha99b]. We argue that appropriate quantitative process models must be considered in such schemes. Based on present authors' work of several years investigating aspects of the software process, this paper summarises experiences and provides guidelines for the building and use of quantitative models.

[1] Shared properties of all *E*-type systems include a loosely defined requirement or expectation that stakeholders are satisfied with the system as is. An *E*-type system is judged by the results it delivers as they relate to the real world, its performance, its behaviour in execution, the functionality it delivers, its ease of use and so on. They display an intrinsic need to be adapted in a continuing basis (that is evolved). In this they differ from *S*-type systems where the criterion of acceptability is that of correctness. The latter, by definition, must be correct, in a mathematical sense, relative to an absolute specification [leh85].

2. Feedback in the Software Process

In 1993 one of the authors asked himself the question: why in spite of the continuing emergence of new software engineering approaches and techniques, such as object orientation, CASE tools and process modelling, is it still generally so difficult to achieve sustained software process improvement? He immediatly realised an explanation that was consistent with observations initially made as early as 1972 [bel72]. Software processes, apart from the most primitive, are complex multi-loop, multi-agent, multi-level *feedback*[2] systems [leh94]. The above mentioned difficulty becomes clear when one considers some of the properties that feedback systems are likely to display. For example:

- exhibit global, that is, externally observable dynamic behaviour
- negative feedback tends to stabilise system behaviour
- positive feedback tends to induce growth and may induce instability
- composite effect of positive and negative feedback is a complex function that involves path characteristics, gains and delays
- impact of forward path changes on system with negative feedback is localised, and not likely to have impact on global system behaviour
- major change of system behaviour requires changes to feedback loop mechanisms
- most external loop mechanisms may have a dominant role determining (and possibly constraining) global behaviour
- influence of individual loop mechanisms in multi-loop system is difficult to determine without appropriate models and simulation tools

The feedback observation was stated as the FEAST hypothesis (*F*eedback, *E*volution and *S*oftware *T*echnology) [leh94], it led to a series of three international workshops [fea93,94], with a coming international workshop planned to July 2000 [fea00] and has been investigated by two successive UK EPSRC funded projects, FEAST/1, from 1996 to 1998 [leh96], and FEAST/2, from 1999 to 2001 [leh98a]. These projects have concentrated in the study of a number of evolving industrial software systems and processes provided by their industrial collaborators: ICL, Logica, Matra-BAe, MOD-DERA, with the recent incorporation of BT (FEAST/2). Lucent Technologies have provided data on two of their real time systems thanks to the good offices of one of the FEAST Senior Visiting Fellows. The FEAST projects have collected evidence that supports of the feedback nature of the software process [fea00]. The results to date have been overall consistent with earlier conclusions reached

[2] Feedback is understood here as "...return of part of the output of a system to its source, esp. so as to modify the output..." Oxford Advanced Learner's Dictionary [oxf89].

during the seventies and eighties [leh85]. Findings and conclusions of the project to date are summarised in a number of publications [fea00]. Though the main focus in FEAST has been on the study of evolving systems, many of the experiences derived and conclusions reached to date have also clear implications in the context of *ab initio* software development. All in all, the evidence to date in support of the feedback nature of software processes comes from several sources:

- study of phenomenology of industrial software evolution processes which reveal the latter as an intrinsic feedback process in which the deployment of a system in the operational domain changes the domain [leh85,94,fea00]

- simulation-based modelling of such processes by system dynamics (SD) [for61] techniques, for example, [cha99a]. Feedback also reflected by simulation work by others (SD and other paradigms[kel99])

- analysis of metric data from evolving software processes [bel72,leh85, leh98c]

FEAST/2 is currently exploring, *inter alia*, ways in which industrial organisations can exploit feedback to achieve externally visible, sustained software process improvement. The role of quantitative process modelling in this regard is considered essential.

Frameworks have been developed to assist organisations to achieve increasingly higher levels of process maturity, as for example the Capability Maturity Model (CMM) [pau93], SPICE, Bootstrap, ISO9001, [ele99,zah97]. These approaches have inherited from and proposed the use [hum89] of concepts from quantitative statistical control [box97]. *Measurement* guidelines have been attached to some of these approaches [zah97]. In the context of software evolution and process improvement, quantitative modelling, considered essential to achieve understanding, has not been widely discussed.

3. Quantitative Modelling

In fact, numerous forms of quantitative models have been proposed to support aspects of the management of software organisations for more than thirty years, as for example, for cost estimation. Quantitative modelling can support many aspects within process improvement. It can help establishing baselines of externally visible key process attributes and subsequent process behaviour monitoring. It can also support software process design and change to achieve the desired improvements.

Managers may decide to design and perform their own empirical studies in their industrial settings. Experiments, for example, as the ones conducted by [por95] may involve significant effort and statistical expertise before eventually reaching valid conclusions. For those, for example, many industrial settings, to whom the rigour of experimentation is beyond their resources, due to schedule, cost and other constraints, *lighter weight* but still meaningful methods are needed.

As mentioned in previous sections, two types of quantitative models are relevant to the present discussion. A quantitative model may reflect aspects of the internal mechanisms and elements of the structure of the process being modelled (white-box). Alternatively, as depicted in figure 1, only external behaviour as defined by the lines crossing a set of boundaries, is studied (black-box)[3]. White-

[3] For the interested reader, a discussion of formal aspects of black-box and white-box modelling is given in [kap94].

box model building usually requires detailed knowledge of the process being modelled and its structure. Black-box models can be generated directly from input-output data without involving significant knowledge about the system being modelled.

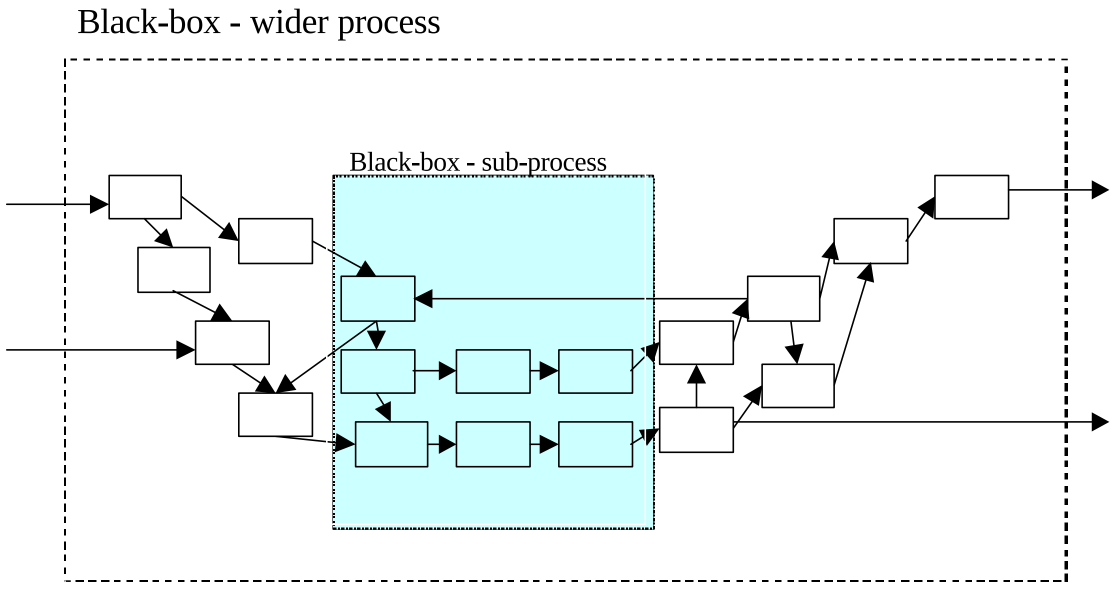


Figure 1 - Boundary Setting for Black Box Models

Within the software engineering and information systems communities, black-box and white-box modelling of the software process have been, in general, considered two independent subject matters and pursued mainly by different communities. The first class, being primarily addressed by the software metrics community, as exemplified by the work reported in a series of International Symposia in Software Metrics [met99]. The second class has been pursued by software process simulation teams, such as the work reflected by the International Software Process Workshop [isp96], initiated by one of the present authors, and by the ProSim Workshops on Software Process Simulation and Modelling [pro00]. Reports on the building and use of black-box and white-box models tend to treat them in an unrelated manner. It has been, however, recognised that there are important connections between the two categories. For instance, white-box models have been proposed as a useful tool for the identification and definition of indicators and metrics, for example in [huf96]. The metrics therefrom derived have a *systemic* character [chr99]. The present authors believe that the use of both types of modelling can be used successfully to address important aspects of the feedaback nature of the software process.

4. Black-Box Modelling

In the FEAST approach black-box model are used to characterise global process behaviour, to establish baselines and determine whether a change in behaviour has happened or not. Main focus of this activity is, for a set of key process attributes expressed as series of numeric values, to characterise their historical behaviour in terms of the following criteria:

- Are definite trends clearly recognisable in the data series?

- Have changes in, for example, average or variance occurred? If yes, when?

- Is the attribute in *state of control* [box97]?

In general the steps involved in black box modelling are the following:

- Metrics definition
- Data smoothing and filtering
- Determine typical patterns, trends, rates and rates of change in the attributes
- Change detection
- Validation

Under limited resources for data extraction and to maintain a discipline, it is advisable to start data collection in a *top-down* fashion. That is, beginning with a small set, probably not more than five or ten, different attributes or metrics that reflect global, externally observable, process behaviour, in terms, for example, of product size, quality and schedule. In FEAST, a suite of metrics of software evolution [leh98b], derived from previous studies have been used with success to characterise and model software process evolutionary behaviour. In fact, such suite of metrics is derived from the attributes initially investigated in the seventies and that eventually yielded a set of eight behavioural patterns, the Laws of Software Evolution [leh74,85,98c].

Once having started with a set of behaviuoral metrics, the list can be progressively refined as the iterative modelling [zur67] cycle progresses and insight is gained. It is recommended to build data sets which reflect behaviour, when possible, over a minimum of six or ten successive releases or time intervals, whichever is available. In addition to a suitable time unit (months, quarters, years) it is also recommended the use of a sequence of releases as an alternative abcissa, since the latter displays usable features, for example, in planning of subsequent releases[fea00].

The next step consist in the search for appropriate models that reflect patterns and regularities in the data, and to determine typical patterns, trends, rates and rates of change in the attributes. Techniques such as the ones used in FEAST [fea00] or their equivalent can be applied (see example in next section).

A number of data filtering or smoothing procedures has been proposed to be used in the treatment of noisy software engineering data [tes98]. While trends may eventually emerge it may be difficult to the unassisted eye to determine change points, in particular if the data is particularly noisy or the changes are small or must be promptly recognised. In this regard, approaches such as CUSUM [bas93,box97] can be applied. For a discussion on metrics validation see, for example, [kit95].

4.1 An Example of Black Box Modelling in FEAST

As an example of black box modelling we will briefly refer here to a recent case study addressing the relationship between effort and software evolution. Cost estimation approaches generally involve *lines of source code* (LOC) or *function points* (FP) as predictors. Our experience is that such metrics are less widely available than might be thought from references to LOC and FP derived productivity and other data in the literature. Metrics can often be extracted from, for example, configuration management databases and change-log records as often kept in industry. One suite of software evolution metrics, that evolved from 70s and 80s studies [leh85], is based on modules and sub-systems and includes the following where the full descriptors in the listing that follows must be read as 'Cumulative number of <descriptor> over a given time interval or per release':

ModifHandlings - changes to modules
ModsChanged - modules changed
ModsCreated - modules added
ModsHandled - modules added or modified
SubsysChanged - subsystems with changed modules
SubsysHandled - subsystems with handled modules
SubsysInclCreations - subsystems with added modules
TotalHandlings - ModulesCreated + ModifHandlings

Space limitations prevent their detailed definitions which will, in any event, vary according to the particular systems or organisation. For this study metric extraction scripts that parse automatically tens of thousands of change log entries and derive the above indicators have been developed in Perl. They also approximate 'effort applied' from analysis of developer identifiers. Similar scripts that extract data from change records, may enable the characterisation of process performance over may years of system evolution , when it is believed that no records are available. Current scripts are customised to individual systems and success requires that records satisfy a degree of regularity in format. Generic procedures are foreseen.

In order to assess the metrics as predictors, a study of the ICL VME operating system kernel is being pursued. It is based on monthly data reflecting 17 years of system evolution. This study has shown that current linear models need re-calibration when the 'effort applied' level changes significantly. We report here on the first of two periods distinct in this respect. 6 linear models were calibrated from the first 5 years. The baseline measures and six models considered in this case study are as follows, with t indicating the month number and *A,B,A1,A2* the models' parameters:

M1 - Baseline 1: *Effort(t) = Average effort over training set*
M2 - Baseline 2: *Effort(t) = [Avg. productivity over training set expressed as (Effort/ModsHandled)] x ModulesHandled(t)*
M3 - *Effort(t) = A.ModsHandled(t) + B*
M4 - *Effort(t) = A.SubsysHandled(t) + B*
M5 - *Effort(t) = A.TotalHandlings(t) + B*
M6 - *Effort(t) = A1.ModsCreated(t)+A2.(ModsChanged(t)-ModsCreated(t))+B*
M7 - *Effort(t) = A1.ModsCreated(t) +A2.(ModifHandlings(t) - ModsCreated(t))+B*
M8 - *Effort(t) = A1.SubsysInclCreations(t) +A2.SubsysChanged(t) + B*

The predictive power of the resultant models was then tested on data from the remaining 5 years. *Mean magnitude of relative error*[4] of approx. 20% was obtained for 5 of the 6 models. Models based on 'subsystems counts' provided a predictive power very similar to those based on finer granularity. The application of these results to effort estimation are encouraging but further studies are needed to establish their wider validity. The above study exemplifies a typical black box modellling of software evolution processes. For the interested reader further details are given in [ram00]. Another interesting example of black box modelling in the context of evolution processes is given in [ysi99].

[4] Defined as Average of (|Predicted Size – Actual Size| /Actual Size)

5. White Box Modelling

White-box model can be built using, for example, simulation-based paradigms such as system dynamics, discrete-event and others [kel99]. White-box models support reason about the structure of the process of interests and related policies. They are appropriate to investigate what are the relevant mechanisms in an existing or in a to-be process, and what changes must be performed in order to achieve the desired behaviour, or alternative, what behaviour will derive from a set of potential changes. They are mainly suggested here as a means to assess the potential impact of process improvements before they are implemented in an existing process. A general discussion on simulation models in the context of the software process is provided in [kel99]. General techniques for the validation and establishing the credibility of simulation models are given in [sch80].

For pointers to the FEAST dynamic models see, for example, [cha99a,fea00]. FEAST models are remarkably simple when one compares them with other SD models reported in the literature. They provides a high level, but yet meaningful view of the process as judged by those who were involved from the industrial collaborators. The FEAST approach to SD restricts detail to the minimum necessary and incorporates elements external to the inmediate technical software process, following in this regard, Zurcher and Randell [zur67] methodology. Participation and agreement of key process personnel involved in the validation of the model (what is called *face value* [kel99]) and in identification of process improvements is considered essential.

5.1 An Example of White Box Modelling in FEAST

Figure 2 presents a simplifed view of a software evolution process using the Vensim *system dynamics* tool [ven95]. The SD modelling technique was initially developed by Forrester [for61]. In a nutshell, a SD model may be seen as a hydraulic system with tanks (termed levels or stocks) and flows between them. The latter are determined by rate variables. There are also variables which are neither levels nor flows. These are called auxiliary variables. Flowrates may be influenced by some of the other variables. Clouds symbolise infinity supplies. Formally, a SD model represents a nonlinear system of differential equations. Modelling tools such as Vensim [ven95] include numerical integration algorithms that solve the equations and produce the simulation results in generally small time steps, in comparison to the simulation interval, that resemble continuous time. SD modelling is also known as continuous simulation to distinguish it from techniques based on discrete time steps [kel99].

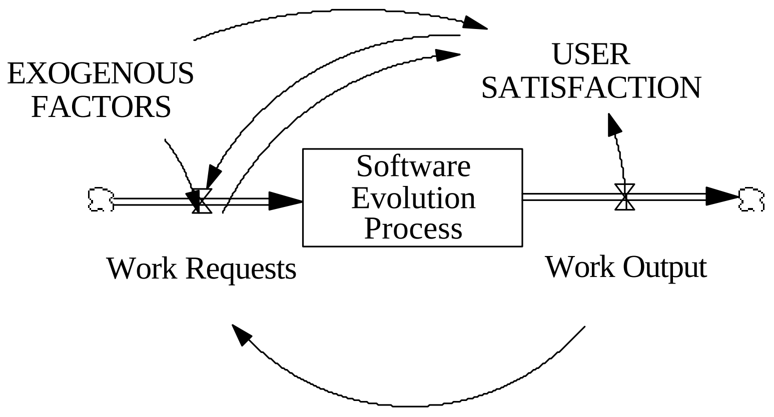


Figure 2 - A Simplified View of a Software Evolution Process

The text in figure 2 represents variables, that may be either flows (the text close to the valve symbols), levels (text within boxes) and auxiliary variables. The double-lined arrows containing valves in the middle represent flows. The single-lined arrows represent "influence" relationship, that is, 'User Satistaction' -> 'Work Requests' indicates that in this model the latter is a mathematical function of the first and of the other variables connected by similar arrows pointing to 'Work Requests'.

Figure 3 displays a next step in modelling refinement. While variables such as 'User Satisfaction' do not appear and are left for further requinement, the model includes further detail and is fully executable in Vensim. Though the present model has not been calibrated to industrial data, it is included here to illustrate the FEAST approach to date to dynamic modelling. The present model addresses the role of *anti-regressive* work policy [leh74,85], as explained below, and contains elements from previous models [rio77] as well as new, directly derived from FEAST experiences [cha99a].

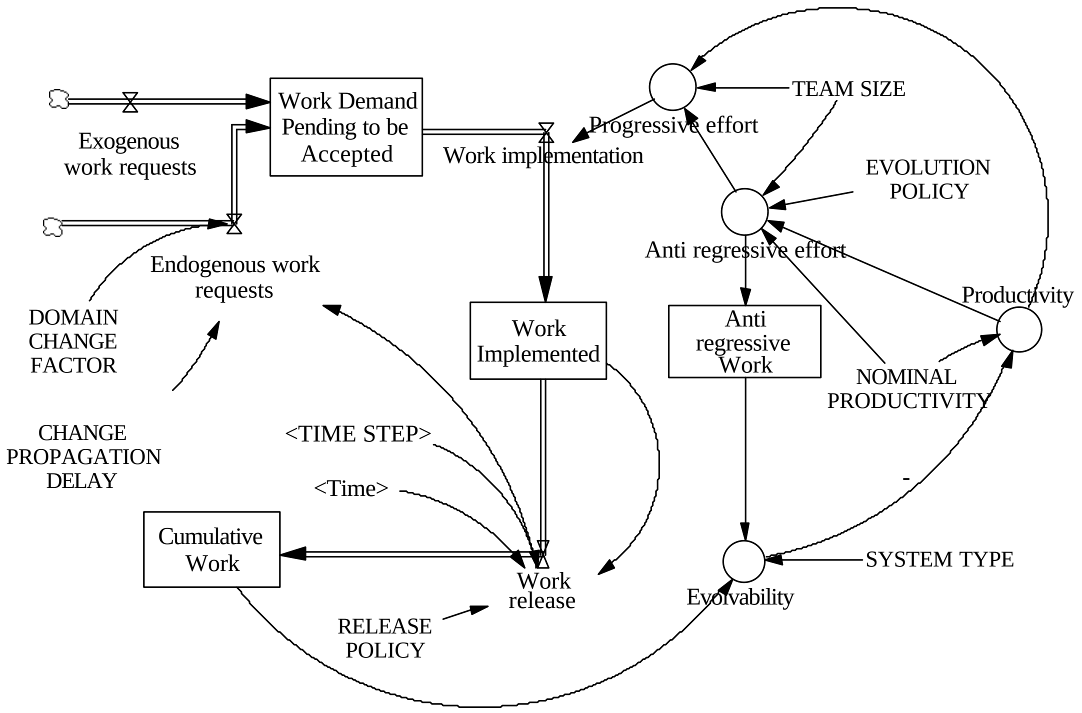
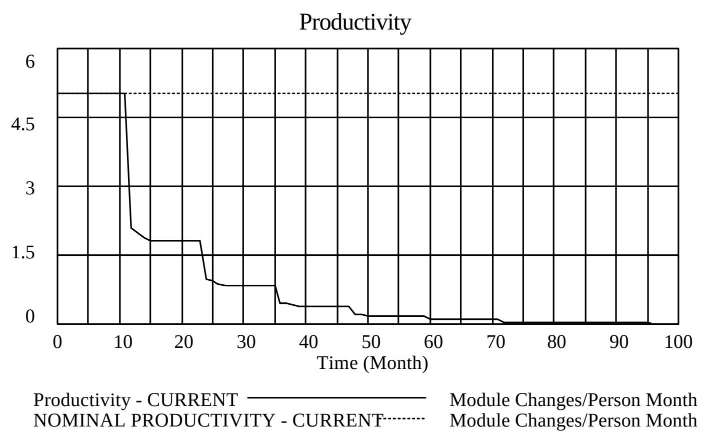


Figure 3 - The Dynamic Model to Investigate the Role of Anti-Regressive Policies

In general software evolution encompasses activities that include fixing, enhancement, adaptation, addition and removal. Lehman suggested a two-tier classification: *progressive* and anti-regressive activities [leh85]. The activities that enhance (by addition or modification) system's functionality were termed progressive. Anti-regressive were those intended to compensate for the effects on the software product of change upon change upon change, such as growing complexity and other software aging effects, for example. In the context of effort, anti-regressive activities represent those activities which consume effort without having an external effect on the software. Hence, they are difficult to justify.



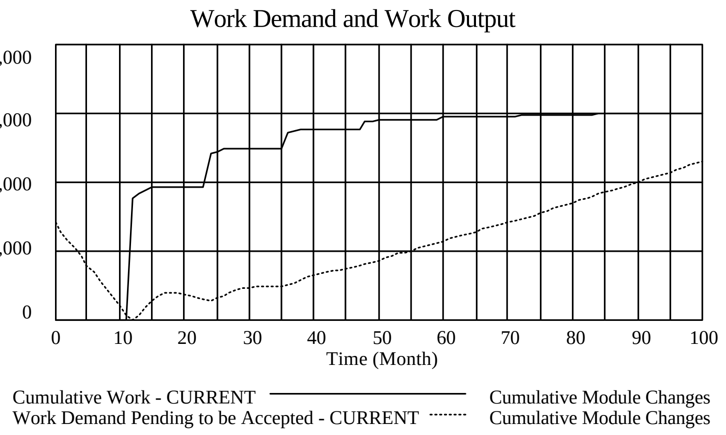
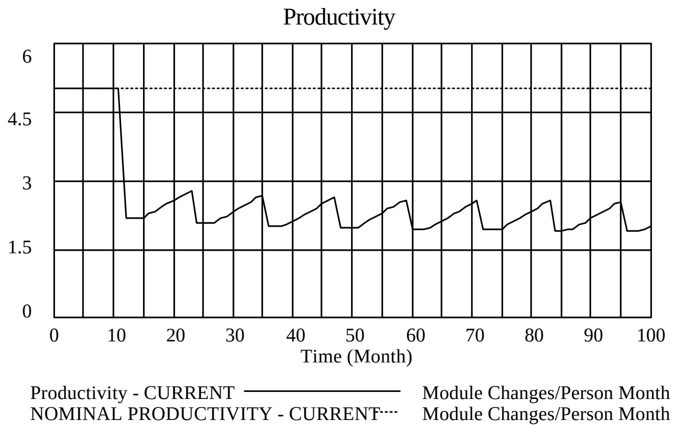


Figure 4 - Simulated Process Behaviour with No Resources Assigned to Anti Regressive Work

The model in figure 3, for example, enables us to examine the effect of different evolution policies. In the model, 'Evolution Policy' represents the proportion of the team size assigned to anti-regressive work. It may take any value between 0, that is, no resources assigned to anti-regressive work and 1 corresponding to the totality of resources applied to that activity. The model includes a control mechanism that activates the anti-regressive work when a decrease in productivity with respect to the nominal value occurs. Figure 4 and figure 5 show how sensititive the evolutionary behaviour is to such policy. In particular, figure 5 shows how 30 % of resources assigned to anti-regressive work will expand significantly the period within which evolution is feasible.



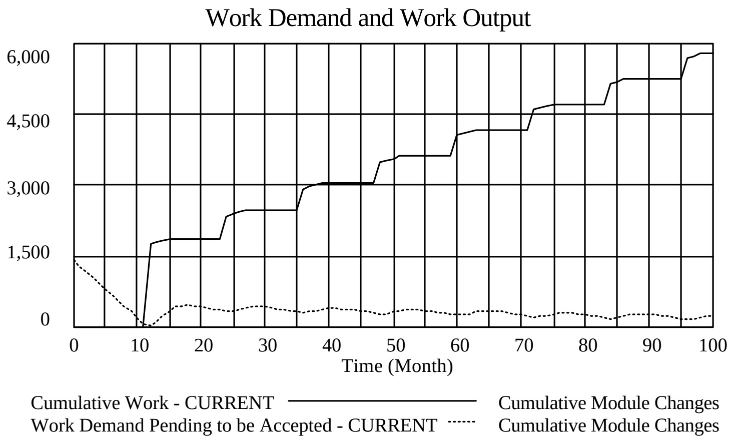


Figure 5 - Simulated Process Behaviour with 30 percent of the Resources devoted to Anti Regressive Work

6. The FEAST Approach

Process iteration, the repeated execution of process steps, however the latter may be defined, appears to be a property of the engineering of software, at least as we currently know it. Iteration occurs at the developer's level, as for example when a piece of code is modified and recompiled once and again in search for the desired behaviour. It also occurs at the software organisation level, when, for example, a new version of the software is generated to replace an older one. Based on present authors experience, black-box and white-box techniques display many complementary features and can be used in a iterative fashion. Figure 6 displays a suggested software process modelling cycle with alternating phases of black-box and white-box modelling. The scheme appears to be particular appropriate in the context of software that is evolved in a continuing basis, as reflected, for example in a sequence of software releases. The generation of each software release offers the opportunity to complete one modelling cycle and to achieve further refinement and validation [kit95,sch80] of the models.

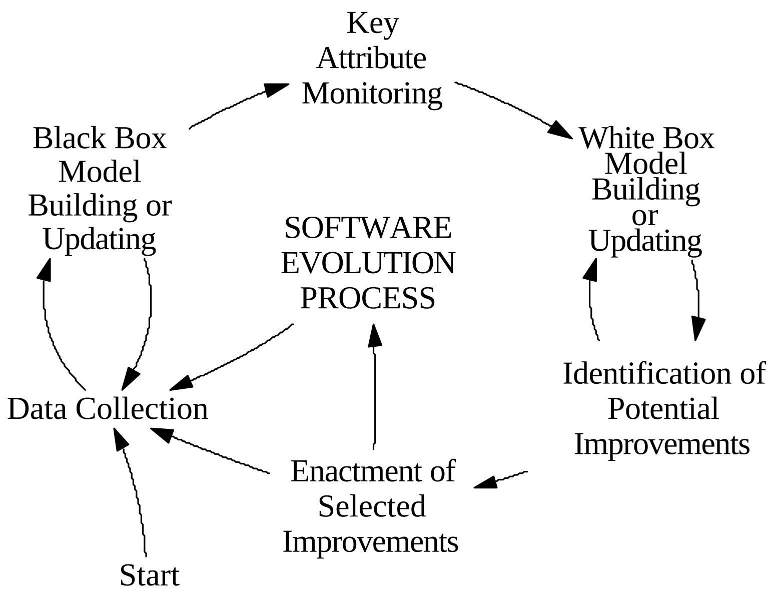

Key Attribute Monitoring
White Box Model Building or Updating
Black Box Model Building or Updating
SOFTWARE EVOLUTION PROCESS
Identification of Potential Improvements
Data Collection
Enactment of Selected Improvements
Start

Figure 6 - Iterating Black-Box and White-Box Modelling

In summary, black box models are used to generate baselines and in further cycles, also to assess the effectiveness of the improvements. White box models serve primarily as a basis for identification of potential improvements. What-if experiments and sensitivity analysis are performed, for example, to assess various changes in forwards and feedback paths. The simulated behaviour under diverse risk scenarios. Assumptions are recorded, so that they can be checked thereafter. Once the selected changes to the process have been implemented and when new data is available, black box models are updated as a result of the new observations. This provides an independent assessment of the effects of the process changes being undertaken. White box models can now be updated as required and the cycle repeats.

7. Final Remarks

This paper must be seen as a first attempt to address the issue and limited to the length of a conference paper. It is hoped that as the FEAST/2 progresses the experiences and guidelines will be refined and other relevant aspects documented. For example, the role of quantitative models at a particular maturity level [pau93] was not discussed. The interested reader will find a relevant discussion in [chr99]. Limitations of process models have been discussed, for example, in [leh85, chapter 11]. Even in the light of these and other limitations, they are essential for improving and disseminating understanding of the process, building a common language and sharing of concepts and process views amongst those involved.

One of the barriers to achieve a success in this regard comes from the knowledge burden imposed by the modelling process and modelling tools. It is, however, possible to envisage that quantitative process modelling support systems of the future will offer the practitioner a suite of ready-to-be-used black-box and white-box models and tools for their usage and their combination. Essential requisite for these systems is to be intuitive and friendly but also sophisticated enough not to impose an excessive knowledge-burden on the model builder and process improvement analyst.

8. Acknowledgements

Many thanks are due to the FEAST industrial collaborators for providing data and for discussions that have been essential for the progress made. Grateful thanks are also Profs. D Perry and W Turski, Senior Visiting Fellows at the Dept. of Computing, Imperial College, and also to Dr. Paul Wernick, former FEAST/1 team member, for many helpful discussions. This work has been financially supported by the UK EPSRC, through the FEAST/2 project, grant number GR/M44101.

9. References

[bas93] Basseville M and Nikiforov IV, *Detection of Abrupt Changes: Theory and Application*, PTR Prentice Hall, Englewood Cliffs, NJ, 1993, 528 pps

[bel72] Belady LA and Lehman MM, *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 [leh85].

[boe81] Boehm B, *Software Engineering Economics*, Englewood Cliffs, New Jersey, Prentice-Hall, 1981, 767 pps.

[box97] Box G and Luceño A, *Statistical Control by Monitoring and Feedback Adjustment,* Wiley, New York, 1997, 327 pp

[cha99a] Chatters BW, Lehman MM, Ramil JF, Wernick P, *Modelling a Software Evolution Process*, ProSim'99, Softw. Process Modelling and Simulation Workshop, Silver Falls, Oregon, 28-30 June 99, to appear as *Modelling a Long Term Software Evolution Process* in Software Process - Improvement and Practice, in 2000

[cha99b] Chatters BW, *Implementing an Experience Factory: Maintenance and Evolution of the Software and Systems Development Process*, Proc. Intern. Conf. on Software Maintenance, ICSM'1999, 30 Aug. to 3 Sept. Oxford, UK, pp. 146 - 151

[cla98] Clark B, Devnani-Chulani S and Boehm B, *Calibrating the COCOMO II Post-Architecture Model*, Proc. ICSE'20, April 19-25, Kyoto, Japan, 1998, pp. 477 - 480

[chr99] Christie A, *Simulation in support of CMM-based Process Improvement*, The Journal of Systems and Software Vol. 46, Nos. 2/3, 1999, pp 107 - 112

[ele99] El Eman K and Madhavji NH (eds.), *Elements of Software Process Assessment and Improvement*, IEEE CS Press, 1999

[fea94,5] *Preprints of the three FEAST Workshops*, M M Lehman (ed.), Dept. of Comp., ICSTM, 1994/5. Available from links at [fea00]

[fea00] *FEAST, Feedback, Evolution and Software Technology*, FEAST project web page, http://www-dse.doc.ic.ac.uk/~mml/feast

[for61] Forrester JW, *Industrial Dynamics*, MIT Press, Cambridge, Mass., 1961

[huf96] Huff KE, *Process Measurement though Process Modelling and Simulation*, in the proceedings of [isp96], pp. 97-99

[hum89] Humphrey WS, *Managing the Software Process*, Addison-Wesley, 1989

[isp96] Proceedings of the 10th International Software Process Workshop "Process Support of Software Procuct Lines", IEEE Computer Society, June 17 - 19, 1996, Ventron, France.

[kah00] Kahen G, Lehman MM and Ramil JF, *Model Based Assessment of Software Evolution Processes,* Res. Rep., Dept. of Comp., Imp. Col., London, Feb. 2000, 11 pps.

[kel99] Kellner MI, Madacy RJ and Raffo DM, *Software Process Simulation Modelling: Why? What? How?*, Journal of Systems and Software, Vol. 46, No. 2/3, April 1999, pp 91 -106

[kit95] Kitchenham B, Pfleeger SL and Fenton N, *Towards a Framework for Software Measurement Validation*, IEEE Trans. on Softw. Eng., Vol. 21, No. 12, Dec 1995, pp 929 - 944

[kun98] Kung HJ and Hsu C, *Software Maintenance Life Cycle Model*, Proc. Int. Conf. Softw. Maintenance, Bethesda, Maryland, Nov. 16-20, 1998, pp 113 - 121

[leh74] Lehman MM., *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 *Programming Methodology*, (D. Gries. ed.), Springer Verlag, 1978, pp. 42-62.

[leh85] Lehman M.M. and Belady L.A., *Software Evolution - Processes of Software Change*, Academic Press, London, 1985, 538 p.

[leh94] 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 Softw. Technology, sp. is. on Software Maintenance, vol. 38, no. 11, 1996, 681 - 686

[leh96] Lehman MM, *FEAST/2: Case for Support*, Department of Computing, Imperial College, London, UK, Jul. 1998. Available from links at the FEAST project web site [fea00].

[leh98a] Lehman MM, *FEAST/2: Case for Support*, Department of Computing, Imperial College, London, UK, Jul. 1998. Available from links at the FEAST project web site [fea00].

[leh98b] Lehman MM, *et al*, *Implications of Evolution Metrics on Software Maintenance*, Proc. Int. Conf. on Soft. Maint. ICSM'98, Bethesda, MD, 16 - 18 Nov. 1998, pp 208 - 217

[leh98c] Lehman M.M., Perry D.E. and Ramil J.F., *On Evidence Supporting the FEAST Hypothesis and the Laws of Software Evolution*, Proc. Metrics'98, Bethesda, Maryland, Nov. 20-21, 1998

[met99] *Proceedings of Metrics 1999* - Sixth International Symposium on Software Metrics, November 4-6, 1999, Marriott Hotel, Boca-Raton, Florida, USA

[pau93] Paulk MC et al, *Capability Maturity Model for Software, Version 1.1*, Software Engineering Institute Report CMU/SEI-93-TR-24

[por95] Porter A, Siy H and Toman CA and Votta LG, *An Experiment to Assess the Cost-Benefits of Code Inspections in Large Scale Software Development*, SIGSOFT 95, Washington, pp 92-103, 1995

[pro00] *Call for Papers and Participation*. ProSim 2000, Third Workshop on Software Process Simulation and Modelling, July 12-14, 2000, Imperial College, London, UK,

[ram00] Ramil JF and Lehman MM, *Metrics of Software Evolution as Effort Predictors - A Case Study*, submitted for publication, Feb. 2000

[rio77] Riordan JS, *An Evolution Dynamics Model of Software Systems Development*, in Software Phenomenology - Working Papers of the (First) SLCM Workshop, Airlie, Virginia, Aug 1977. Pub ISRAD/AIRMICS, Comp. Sys. Comm. US Army, Fort Belvoir VI, Dec 1977, pp 339 - 360

[sch80] Schruben LW, *Establishing the Credibility of Simulations*, Simulation, 34, 1980, pp 101- 105

[siy99] Siy H and Mockus A, *Measuring Domain Engineering Effects on Software Coding Cost*, Metrics 1999 - Sixth Intern. Symp. on Software Metrics, November 4-6, 1999, Boca-Raton, FL, USA

[tes98] Tesoreiro R and Zelkowitz M, *A Model of Noisy Software Engineering Data, Status Report*, Proc. ICSE'98, April 19-25, 1998, Kyoto, Japan, pp 461 - 476

[yas97] Yasuhiro Mashiko and Victor R. Basili, *Using the GQM Paradigm to Investigate Influential Factors for Software Process Improvement*, J. of Syst. and Softw., Vol. 36, No. 1, pp 17-32, Jan 1997

[zah97] Zahran S, *Software Process Improvement - Practical Guidelines for Business Success*, SEI Series in Software Engineering, Addison-Wesley, Harlow, England, 1997, 447 pps.

[zur67] Zurcher FW and Randell B, *Iterative Multi-Level Modeling - A Methodology for Computer System Design*, IBM Res. Div. Rep. RC-1938, Nov. 19678. Also in Proc. IFIP Congr. 1968, Edinburgh, Aug 1968, pp D-138 - 142