Thoughts on the Role of Formalisms in Studying Software Evolution

International Special Session on Formal Foundations of Software Evolution, co-located with CSMR 2001
13 March 2001, Lisbon

Meir M Lehman Juan F Ramil Goel Kahen
Department of Computing
Imperial College of Science, Technology and Medicine
180 Queen's Gate, London SW7 2BZ
tel +44 20 7594 8216; fax +44 20 7594 8215
{mml,ramil,gk}@doc.ic.ac.uk

Summary
This paper presents a *system dynamics* model of a long-term software evolution process as an example of process *behavioural formalism* and shows how the model permits assessment of the impact of various policies on evolutionary attributes. The model provides a context and framework within which at least three crucial software management tasks, resource allocation, release planning, and process performance monitoring can be tackled. It is part of and exemplifies the methods for software process modelling being developed and applied in the FEAST, *F*eedback, *E*volution *A*nd *S*oftware *T*echnology, projects.

1 Introduction

The term *software evolution* relates to the activity and phenomenon of software change [leh85]. It includes two aspects that reflect, respectively, the complementary concerns of the *how* and the *what/why* [leh00b] of software evolution. Interest in the former is concerned with *methods*, *tools* and *techniques* to change functional, performance and other characteristics of the software in a controlled, reliable, fast and cost effective manner. This is the more widespread view and is exemplified by the contributions to a series of meetings on Principles of Software Evolution [ispse98,00]. Interest in the *what/why*, on the other hand, focuses on *understanding* the software evolution *phenomenon*, its underlying causes and drivers, common patterns of evolutionary behaviour, and the characteristics of that behaviour. This line of investigation, the focus of the FEAST (*F*eedback, *E*volution *A*nd *S*oftware *T*echnology) studies in the Department of Computing at Imperial College [feast] and their antecedents, has also been pursued by a small number of other groups world-wide [e.g. kem99,coo00,gdf00,raj00].

Both views, the *how* and the *what/why*, must be pursued if mastery of the software evolution phenomenon is to be achieved in a world increasing dependent on computers and software. The following are examples of the type of questions whose answer is pursued under the latter view:

- why does software evolution occur?
- why is it inevitable?
- what are key attributes of the evolution process?
- what is their impact on the software process and its products?
- what are the practical implications of the above on the planning control and management of software system evolution?

One of the present authors (mml) has been actively involved in studies of software evolution for more than 30 years [leh69,85,feast]. This work has resulted in a set of laws of software evolution [leh74,78,80a,b,85,feast], the *SPE* program classification scheme [leh80b], a principle of software uncertainty [leh89,90] and, most recently, a FEAST hypothesis [leh94feast]. The results of the recent work within the FEAST projects are scattered in some 40 papers published since 1996 [feast]. A full listing is available from the project web site http://www.doc.ic.ac.uk/~mml/feast.

2 Towards a Formal Theory of Software Evolution

It is the view of the present authors that formalisms can play as important a role in the study of the *what* and the *why* of software evolution as they do in the *how* view, even though they serve different purposes. In the *how* mode, they are primarily intended to be used as representations of different models of the *application*; that is, specifications, programs, the operational, evolution domains and even of entities such as the evolution process, system architectures and relationships such as abstraction and satisfaction [mai00]. And all these models must permit continual representation of the subject as it evolves. The power of appropriate formalisms in this area is clear.

3 Behavioural Formalisms

One of the roles of formalisms under the *what/why* view can be to facilitate precise reasoning about the behaviour of the evolution process, and its product. Managers and process designers could frequently benefit from reasoned exploration of behavioural issues but lack the reasoning tools to do so. Of equal relevance is the potential role of formalisms in guiding the direction and likelihood of future changes in process, product or domain attributes or the direction and likelihood of future changes in *needs*.

Formalisms to facilitate such reasoning have emerged, for example, from the work in process modelling languages over the last 15 years or so [ost87,97,pot97]. The emphasis of that work has been primarily on process *description* and *prescription*. Formalisms have also been applied by the workflow community [e.g., wir00]. Combining both concepts, models such as *process programs* [ost87,97] indicate the steps that constitute a process, workflow controls, conditions to activate sub-processes and so on. Within this view, fine-grained characteristics and properties such as absence of deadlock, were also of interest.

The present authors believe that reasoning about process *behaviour* and about properties such as the *economic feasibility* of a process or about its quality and other performance, however measured, is at least as relevant as is reasoning about process *description* and *prescription*.

The introduction of formalism to the study of process *behaviour* raises many issues[1]. Some of these have previously been analysed, for example, by those investigating the use of mathematics in sociology [col64]. This brings with it the question whether process behaviour is predominantly *indeterministic* (as defined by Chapman [cha96]) and therefore not, since it involves humans at all levels, in general amenable to mathematical formalisation. The same question arises in the study of the software process. If this view were to prevail, the use of formalisms to study process behaviour would be a futile exercise. Some software process behaviour has, however, been captured in empirical generalisations (e.g. laws of software evolution [leh74,78,80a,b,85,feast]) as has, for example, software process effort estimation in COCOMO [boe00] These are, by themselves, sufficient to demonstrate that there is a role for formalism in the study of process behaviour. Other evidence also derived from empirical studies [e.g. abd91,leh98] supports this conclusion; has demonstrated that mathematical formalisms such as differential equations for example, have their uses in other such studies.

One of the outcomes of the FEAST projects has been the realisation that one may extend the use of formalisms to achieve rigorous representations of behavioural invariants and empirical generalisations such as the laws of software evolution, on the one hand, and rules and guidelines [leh00a] for project management, on the other [leh00c]. If this can be successfully achieved one will be able to provide a formal rationale for what is termed *good practice*. Even more importantly, one will be able to provide a formal theory of software evolution as the foundations for a unified and coherent framework for software engineering. The development of such a theory is the theme of a recent project proposal [leh00c,d].

[1] See [mcg97] for a justification of software process behavioural formalisms from a different but complementary perspective.

4 Software Process Simulation Modelling

The argument in favour of behavioural formalisms accords with a recent call for software engineering research to abandon the flatland of purely *technical* issues and to proceed to incorporate other dimensions such as cost and value [boe00]. One possible way to achieve this and to proceed to a disciplined study of process behaviour is by means of simulation languages and tools, and models derived therefrom. One example of this approach is provided by the work of the process simulation community [kel99,prosim00][2]. Another example is provided by the FEAST projects with its models reflecting aspects of long-term evolution management [feast]. That work involved development of *system dynamics* models [for61,abd91,coy96]. The tool used was *Vensim*® [ven95].

As briefly discussed in section 7, those models provide an example of the use of formalisms, in this case system dynamics, for the study software evolution from the behavioural point of view. The work is illustrated here by a model intended for use in long-term planning and management of software evolution processes. The outputs of this model all relate to the evaluation of effort allocation policies. However, alignment of the present model to actual industrial processes, its calibration against them and determination of its domain and extent of validity [for80] remain to be done. If successfully accomplished, the result will be a model that can be used within the processes it reflects for their further planning, management and improvement.

Incidentally, this application draws attention to an issue considered in other disciplines and specifically addressed by Heisenberg's Principle of Uncertainty. Using a model of system evolution to plan implementation of that evolution will influence resultant process behaviour, is indeed intended to do so. Thus, it may serve as a self-fulfilling prophecy, confirming (and perpetuating) the validity of the model, even though objectively it does not accurately reflect the phenomenon. This observation appears to point to a fundamental principle relating to the evolution process. It cannot be pursued here other

[2] Formalisms such as Petri-Nets [e.g. kus97] or state charts and the *STATEMATE*® system [e.g. har90] have also been used in process behavioural modelling. We do not here discuss under what circumstances one formalism is more appropriate than another in this application or whether a combination of formalisms can offer an advantage [ram98].

than to observe that it is related to the observation that software operating in and with a real-world domain incorporates a model of itself [leh85].

In any event, what can be said is that the system dynamic models referred above incorporate behavioural formalisms of software evolution. Hence they are relevant, and hopefully, of interest to FFSE.

5 An Example: Change and Complexity in Evolving Software

Software evolution may be described as the achievement of disciplined software change. It is driven, *inter alia*, by the need to maintain user satisfaction within a changing application and usage domain. Changes are inevitably in the application domain, user familiarity, needs and domain properties. They result from user learning, familiarisation and other developments within an environment in which market forces, human interest and ambition, technology, the influence of factors and agents exogenous to the application and system also play a role. Evolution entails adaptation of existing properties, functionality in particular, and the addition of new capability. The latter implies system functional growth over time and releases. The ultimate goal is to at least maintain and, generally, to increase stakeholder satisfaction.

In the above context, one underlying fact of life must be accepted. As a consequence of the superposition of change upon change upon change, the *complexity* of software systems tends to increase as they evolve [leh74]. Such increase brings with it, pressure for a decline in the attainable *functional growth rate* [e.g. leh98]. Managers can either ignore this decline and face the inevitable consequences of eventual system stagnation. Alternatively they can take cognisance of the complexity growth and divert effort to *control* it and any other forces causing the decline in growth rate. Given an awareness that growth trends that constrain system evolution develop, they may well accept the need to direct effort to activities that might otherwise have been overlooked or neglected. However, if the need is not recognised or not accepted such anti-regressive activities will tend to be neglected. This, despite the fact that, unless controlled, as the system evolves, growing complexity will force down system maintenance, adaptation and extension productivity and system quality will deteriorate. This is a fact that cannot be permitted to materialise when control and mastery of system evolution is vital in a society increasingly reliant on *inventories* of ageing software.

6 Complexity Control: Anti-regressive Work

Growing complexity is reflected by increased size, more *interdependent* functionality, a larger number of integrated components, more control mechanisms, a higher level of reciprocal interdependency. It is reflected in and evidenced by greater inter-element connectivity and more complex (sic) interfaces. In this context, the achievement of a minimum level of complexity management and control activity is essential to maintain the rate of system evolution at the desired or required level.

Motivated by Baumol's classification [bau67] of activity into *progressive* and *anti-progressive* types, Lehman suggested [leh74] a further category, *anti-regressive*. Activities that, by addition or modification of functionality for example, enhance system value were termed *progressive*. Effort such as complexity control or reduction, on the other hand, does not, from the short-term point of view, contribute to the perceived value, as reflected, for example, by system functional power or performance. What it achieves is to prevent system decline. If this trend is not controlled, the cost and fault proneness of system evolution will grow; will ultimately constrain system evolution and, in a continually changing world, reduce its value or even render it valueless. This class of activity was termed *anti-regressive*. All effort that compensates for *ageing* effects is included in this class. Such work consumes effort without *immediate* visible stakeholder return. What it achieves is to facilitate continued evolution, more easily, more quickly, more reliably and with less effort. It preserves the opportunities for future growth in value.

7 A Model and its Use[3]

The *system dynamics* [for61,80,abd91,coy96,] model presented here has been inspired by the laws of software evolution [leh74,85,feast], fieldwork with FEAST/2 collaborators and a study of how others approached the development of models of the software process.

Originally inspired in the context of mathematical system theory, *system dynamics* (SD) [for61,coy96] and tools such as *Vensim*® [ven95] that implement and support it, was developed to study the behaviour over time (dynamics) of industrial and managerial systems. Its vocabulary, involving terms such as *levels* (or *stocks*), and *flow* (or *rate*) variables, was inspired by hydraulic systems that appeared to offer intuitive appeal. Guidelines for reinterpretation in other domains may be found in [e.g. for61,coy96].

SD's mathematical formalism is that of differential equations. An SD model is essentially a set of non-linear first-order differential equations:

$$d\mathbf{x}(t)/dt = f(\mathbf{x}(t),\mathbf{p})$$

where t represents the real-time variable, $\mathbf{x}(t)$ is a vector of *levels*, $\mathbf{p}$ a vector of parameters and f() is a non-linear vector-valued function. It is particularly powerful for the representation and simulation of

[3] For a more detailed description of the model see [kah00].

systems involving feedback loops and mechanisms and in that context makes heavy use of numerical methods for the integration of differential equations. In the context of systems dynamics the latter are derived from system visualisations as represented in the system dynamics formalism.

At first sight the underlying formal mathematical models would seem inappropriate in the software engineering context. As illustrated by the example that follows the results obtained so far in FEAST [feast], provides a degree of evidence that the approach is useful. It suggests that as a multi-loop, multi-level, multi-agent feedback system [leh94], the long term, global, behaviour of the global software process is primarily determined by its feedback nature, and by *implied* equations as defined by the visualisations. The model, and language used to represent it, constitute a formalism.

The semantics and syntax of system dynamic models and the procedures to build and validate them have been described in many references [e.g. coy96,ven95]. Two different representations are generally used: *influence* and *level-rate* diagrams.

The structure of the behavioural relationships within a software evolution process can be sketched using *influence diagrams* [coy96]. Influences between any two attributes can be either balancing as for negative feedback or reinforcing as for positive feedback. An influence diagram presents the attributes of interest (in pictures and/or text). Arrows represent influences. A "+" character close to the arrow indicates a positive influence, such as "...the higher the variable at the arrow's origin, the higher the attribute at the arrow's end...". A "-" character indicates the opposite influence. This represents a simple, but effective, view of expected relationships.

The influence diagram in figure 1 constitutes a simplified view of the model to be discussed and the influences it encompasses. In the figure, arrows with solid shafts indicate relationships that are definitively positive or negative. Arrows with dashed shafts indicate influences that, under some circumstances may be positive, under others, negative.

Fig. 2 is a *level-rate diagram* representing the full model as developed using the *Vensim*® tool [ven95].

The variables in the boxes represent *levels* or *stocks*. The variables on the valve icons represent *flows* or *rates*. The variables in circles are auxiliary variables. The remaining variables are model parameters. Arrows with double lines represent flows of information or material that are conserved throughout the execution of the model. Clouds represent either sources or sinks of information or material. The Appendix presents the model of figure 2 in the *Vensim*® tool's language [ven65].

This relatively simple model represents the process at a high-level of abstraction, enabling the *global* nature and influences on the process [leh94] to be more easily understood. It is intended to provide a tool for use in the context of planning and management of software evolution. It relates specifically to demonstrating the influence of the progressive to anti-regressive effort ratio on the long term growth rate using the model as in figure 3 as an executing process simulation. A detailed discussion of the plots is not appropriate here, and the plots are presented as results typical of what one would expect when studying process dynamic behaviour.

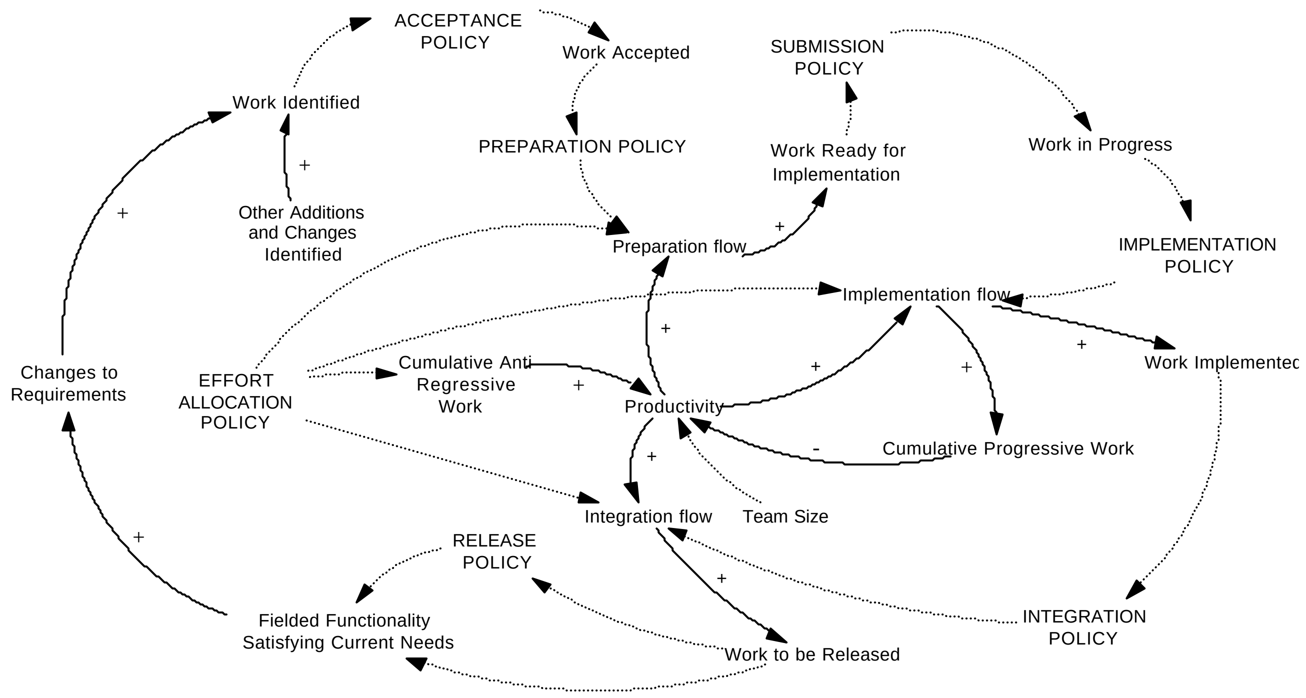

Figure 1: Influence diagram of an ideal software evolution process

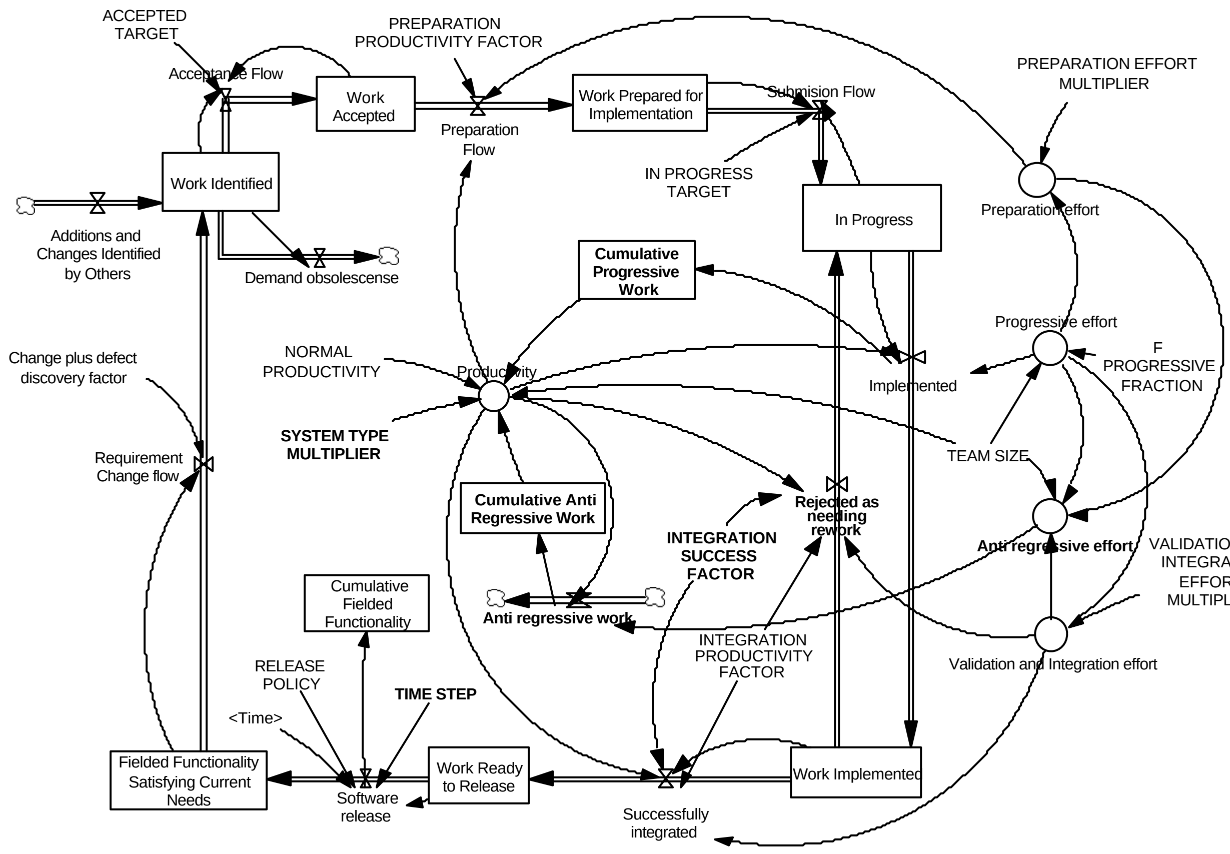
ACCEPTED TARGET
Acceptance Flow
Work Accepted
PREPARATION PRODUCTIVITY FACTOR
Preparation Flow
Work Prepared for Implementation
Submision Flow
PREPARATION EFFORT MULTIPLIER
Work Identified
Additions and Changes Identified by Others
Demand obsolescense
IN PROGRESS TARGET
In Progress
Preparation effort
Cumulative Progressive Work
Progressive effort
F PROGRESSIVE FRACTION
Change plus defect discovery factor
NORMAL PRODUCTIVITY
Productivity
Implemented
SYSTEM TYPE MULTIPLIER
TEAM SIZE
Requirement Change flow
Cumulative Anti Regressive Work
Rejected as needing rework
INTEGRATION SUCCESS FACTOR
Anti regressive effort
VALIDATION AND INTEGRATION EFFORT MULTIPLIER
Cumulative Fielded Functionality
Anti regressive work
INTEGRATION PRODUCTIVITY FACTOR
Validation and Integration effort
RELEASE POLICY
TIME STEP
<Time>
Fielded Functionality Satisfying Current Needs
Software release
Work Ready to Release
Successfully integrated
Work Implemented

Figure 2: Detailed view of the system dynamics model
(no meaning is here to be attached to arrow line thickness)

The plots in figure 3 represent the effects of three different policies that address in particular the level of effort assigned to anti regressive work. Resource available is kept constant throughout. Three policies, AR60, AR40, AR0, are compared. They correspond to the application of 60, 40 and 0 percent of the available effort to anti regressive work. This, in turn, corresponds to 20, 30 and 50 percent, respectively, of the effort available to progressive work as reflected by the variable *F Progressive Fraction.* Remaining resources are shared by the two other activities, *Preparation* and *Validation and Integration.* Simulation results (see fig. 3) show that AR40 leads to higher *Cumulative Fielded Functionality*, than either AR60 or AR0. The accompanying behaviour of other model variables is also presented in the plots. The model offers the basis for other policy analyses relevant, in this instance, to *release planning* [e.g. leh00a]. Moreover, the variables in this model provide a set of attributes that are more generally useful in monitoring and planning evolution process performance.

This workshop, with its focus on formal foundations of software evolution is not the appropriate occasion to enlarge further on the model or to discuss what else may be learned from it with regards to the software process and its products. The brief discussion presented and the principles underlying its development are simply intended to demonstrate the relevance and application of formal methods *in the wider sense.* In this instance the discussion has focussed on the study of the *what*/*why* of software evolution and their potential as tools for the planning and management of long-term evolution processes. Another is provided by the proposal to develop a formal theory of software evolution. The middle ground between purely prescriptive (normative) and behavioural process models remains unexplored. *Semi-normative* theories [col64] may prove to be a useful path to follow for further study of this topic.

8 Final Remarks

This paper suggests that formalisms may not only be relevant in the context of methods and tools to evolve software, that is, the realm of the *how* to achieve software evolution through software change, but also within the investigation of the *what* and *why* of the evolution process.

Our thesis has been that such formalisms, together with models implemented using them, may help in planning and management of long-term evolution. The latter if undertaken, would aim at achieving the above in a reliable, timely and cost-effective way. Its achievement, of course, involves many unsolved challenges. Continuing change and increasing system complexity phenomena, the focus of the simulation model presented, is, however, only one of many influences determining behavioural attributes of long-term software evolution processes and products.

More generally, simulation models developed according to some rigorous discipline may be considered as a formalisation of the software process that provides means to analyse and reason about its behaviour. Other formalisms may be useful for reasoning about and justifying *good practice*. The latter will, we believe, be derivable as theorems from a theory of software evolution to be developed in a project, currently awaiting funding decision [leh00c,d]. That development will be seeded and driven by the behavioural invariants and empirical generalisations observed over the years in the FEAST [feast] and similar studies.

9 Acknowledgements

Financial support from the UK EPSRC, grant GR/M44101 (FEAST/2), is gratefully acknowledged.

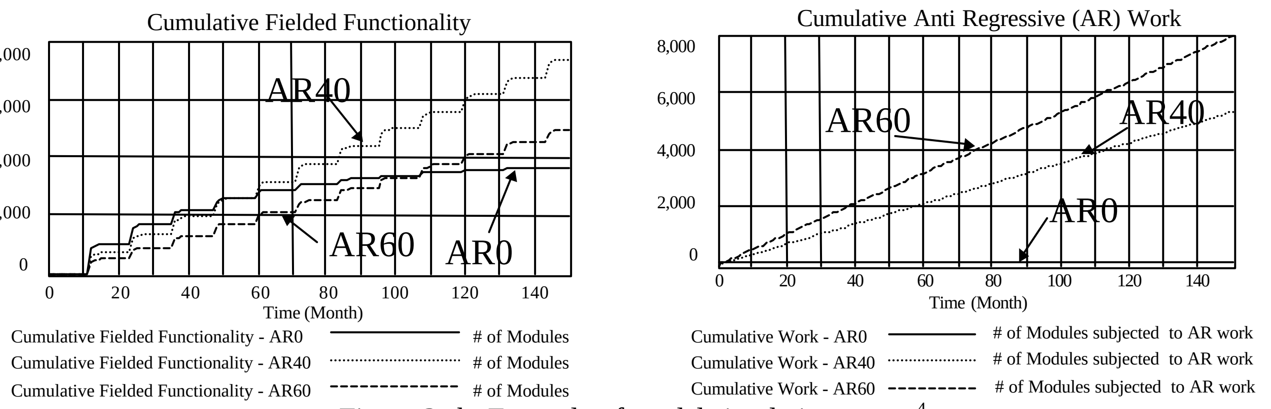

Figure 3a,b: Example of model simulation output[4]

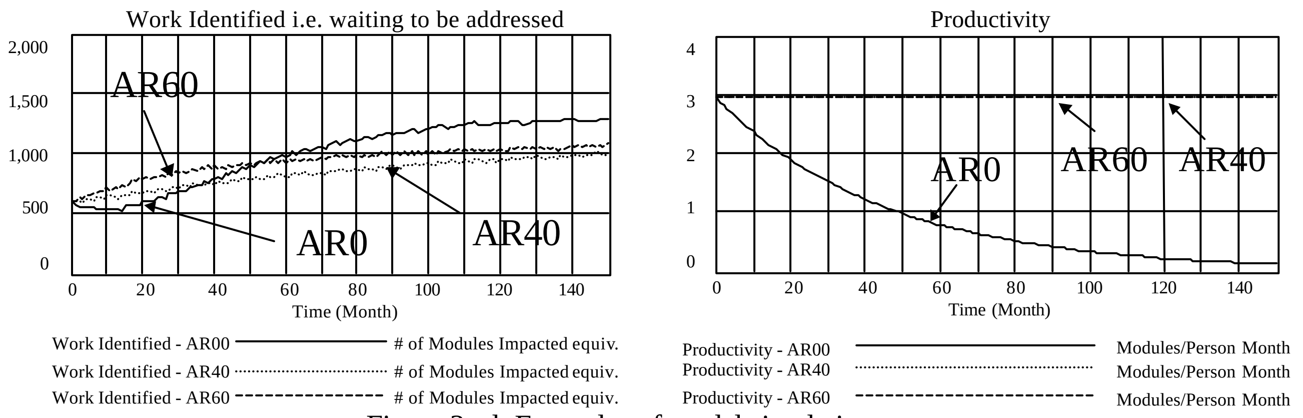

Figure 3c,d: Examples of model simulation output

10 References - An * indicates that the reference has been reprinted in [leh85]. ** indicates that is available from links at http://www.doc.ic.ac.uk/~mml/feast

[abd91] Abdel-Hamid T and Madnick S, Software Project Dynamics - An Integrated Approach, Prentice-Hall, Englewood Cliffs, NJ.

[bau67] Baumol WJ, Macro-Economics of Unbalanced Growth - The Anatomy of Urban Cities, Am. Econ. Review, Jun 1967, pp. 415 - 426

[boe00] Boehm BW and Sullivan KJ, Software Economics: A Roadmap, in Finkelstein A (ed.), The Future of Softw. Eng., ICSE 22, pp. 321 - 343

[cha96] Chatfield C, The Analysis of Time Series - An Introduction, 5th Ed., Chapman & Hall, London, 1996

[col64] Coleman JS, Introduction to Mathematical Sociology, Collier-Macmillan Limited, London, 1964, 554 pp.

[coo00] Cook S *et al*, Software Evolution and Software Evolvability, working paper, U. of Reading, Aug. 2000

[coy96] Coyle RG, System Dynamics Modelling - A Practical Approach, Chapman & Hall London, 413 p

[feast] Feedback, Evolution And Software Technology, Project Web Site http://www.doc.ic.ac.uk/~mml/feast

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

[for80] Forrester JW and Senge P, Tests for Building Confidence in System Dynamics Models, In System Dynamics, Legasto AA Jr., Forrester JW and Lyneis JM (eds.), TIMS Studies in the Management Sciences, v. 14. North Holland, New York, 1980, pp. 209 - 228

[gdf00] Godfrey MW and Qiang T, Evolution in Open Source Software: A Case Study, Proc. ICSM 2000, 11-14 Oct. 2000, San Jose, CA, pp. 131-142

[har90] Harel D *et al*, STATEMATE: A Working Environment for the Development of Complex Reactive Systems, IEEE Trans. on Softw. Eng., v. 16, n. 4, Apr. 1990, pp. 403 - 414

[4] Module counts (e.g., # of files) exemplify one possible unit. *# of Modules subjected to Anti regressive work* may be too simplistic. In a practical case, other units must be used.

[kah00] Kahen G, Lehman MM and Ramil JF, System Dynamic Modelling for the Management of Software Evolution Processes, Research Report 2000/16, Dept. of Comp., Imp. Col., Nov. 2000

[kel99] Kellner MI, Madachy RJ and Raffo DM, Software Process Simulation Modelling: Why? What? How?, J. of Syst. and Software, v. 46, n. 2/3, April 1999, pp 91 - 106

[kem99] Kemerer C and Slaughter S, An Empirical Approach to Studying Software Evolution', IEEE Trans. on Softw. Engineering, v. 25, n. 4, July/Aug. 1999, pp. 493 - 509

[kus97] Kusumoto S *et al,* A New Software Project Simulator Based on Generalized Stochastic Petri-net, Proc. ICSE 19, Boston, May 17 - 23, 1997, pp. 293 - 302

[ispse98] ISPSE 98, International Workshop on the Principles of Softw. Evolution, 20-21 April 1998, co-located with ICSE 98, Kyoto, Japan

[ispse00] ISPSE 2000, International Symposium on Principles of Software Evolution, Kanazawa, Japan, Nov 1-2, 2000

[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, in Imp. Col. of Sc. and Techn. Inaug. Lect. Seri., v. 9, 1970 - 74, pp. 211 - 229. Also in Programming Methodology, Gries D. (ed.), Springer Verlag, 1978, pp. 42 – 62

[leh77]* *id*, Human Thought and Action as an Ingredient of System Behaviour. In Duncan R & Weston Smith M (eds.), Encyclopedia of Ignorance, Pergamon Press, Oxford, 1977

[leh78]* *id*, Laws of Program Evolution - Rules and Tools for Program Management'. Proc. Infotech State of the Art Conf., Why Software Projects Fail, - April 1978, 11/1-11/25

[leh80a]* *id*, On Understanding Laws, Evolution and Conservation in the Large Program Life Cycle, J. of Sys. and Softw., 1980, 1, (3), pp. 213-221

[leh80b]* *id*, Programs, Life Cycles and Laws of Software Evolution, Proc. IEEE Spec. Iss. on Softw. Eng., 68, (9), Sept. 1980, pp. 1060-1076

[leh85] Lehman MM & Belady LA, Software Evolution - Principles of Software Change, Acad. Press, London, 1985

[leh89] Lehman MM, Uncertainty in Computer Application and its Control Through the Engineering of Software, J. of Softw. Maint.: Res. Pract., v. 1, n. 1, Sept. 1989, pp. 3-27

[leh90] Lehman MM, Uncertainty in Computer Application, Tech. Let., CACM, v. 33, n. 5, May 1990, pp. 584-586

[leh94] Lehman MM, Feedback in the Software Process, Keynote Address, CSR Eleventh Annual Wrksh. on Softw. Ev. - Models and Metrics. Dublin, 7-9th Sep. 1994. Also in Info. and Softw. Tech., spec. iss. on Softw. Maint., v. 38, n. 11, 1996, Elsevier, 1996, pp. 681 - 686

[leh98] Lehman MM, Perry DE and Ramil JF, On Evidence Supporting the FEAST Hypothesis and the Laws of Software Evolution, Proc. Metrics'98, Bethesda, Maryland, 20-21 Nov. 1998, pp. 84 - 88

[leh00a]** Lehman MM, Rules and Tools for Software Evolution Planning and Management, pos. paper, FEAST 2000 Workshop, Imp. Col., 10 - 12 Jul. 2000, http://www.doc.ic.ac.uk/~mml/f2000

[leh00b]** Lehman MM, Ramil JF and Kahen G, Evolution as a Noun and Evolution as a Verb, SOCE 2000 Workshop on Software and Organisation Co-evolution, Imp. Col., London, 12-13 Jul. 2000

[leh00c] Lehman MM and Ramil JF, Towards a Theory of Software Evolution - And Its Practical Impact, invited talk, ISPSE 2000, Intl. Symp. on the Principles of Softw. Evolution, Kanazawa, Japan, Nov. 1-2, 2000

[leh00d] Lehman MM, TheSE - An Approach to a Theory of Software Evolution, proj. prop., DoC, Imp. Col., Dec. 2000

[mai00] Maibaum TSE, Mathematical Foundations of Software Engineering: a Roadmap, in A. Finkelstein (ed.), The Future of Software Engineering , ICSE 2000, June 4-11 Limerick, Ireland, pp. 161 - 172

[mcg97] McGrath GM, A Process Modelling Framework: Capturing Key Aspects of Organisational Behaviour, Proc. Australian Softw. Eng. Conf., 1997, pp. 118 - 126

[ost87] Osterweil L, Software Processes Are Software Too, Proc. 9th Int. Conf. on SEng., 1987, pp 2 - 12

[ost97] Osterweil L, Software Processes Are Software Too, Revisited: An Invited Talk on the Most Influential Paper of ICSE 9, Proc. 19th Int. Conf. on Software Engineering, May 17-23, 1987, Boston, MA, pp. 540 - 548

[pod97] Podoroznhy RM and Osterweil LJ, The Criticality of Modeling Formalisms in Software Design Method Comparison - Experience Report, ICSE'97, May 17-23 1997, Boston MA, pp. 303-313

[prosim00] Prosim 2000, Workshop on Software Process Simulation and Modelling, 12-14 July, 2000, Imp. Col., London, http://www.prosim.org

[raj00] Rajlich VT and Bennett KH, A Staged Model for the Software Life Cycle, Computer, Jul.2000,pp.66-71

[ram98]** Ramil JF and Lehman MM, Fuzzy Dynamics in Software Project Simulation and Support, Proc. 6th European Workshop on Softw. Process Technology (EWSPT-6), 16-18 Sept. 1998, Weybridge, UK, LNCS 1487, Springer-Verlag, pp. 122-126

[ven95] Vensim - Ventana Simulation Environment, Reference Manual, Version 1.62, Belmont, MA., 1995

[wir00] Wirtz G, Using a Visual Software Engineering Language for Specifying and Analysing Workflows, IEEE International Symp.on Visual Languages 2000, pp. 97 - 98

Appendix - The Model in Vensim [ven95]

```
Acceptance Flow =
IF THEN ELSE((Work Accepted<ACCEPTED TARGET)
:AND:(Work Identified>0),300,0)
    ~
    ~     |
ACCEPTED TARGET = 100
    ~
    ~     |
Additions and Changes Identified by Others = (RANDOM
POISSON(60))
    ~ Changes/Month
    ~     |
Anti regressive effort =IF THEN ELSE (
(TEAM SIZE-Preparation effort
-Progressive effort-Validation and Integration effort)>0,
(TEAM SIZE-Preparation effort-Progressive effort
-Validation and Integration effort),0)
    ~
    ~     |
Anti regressive work= Anti regressive effort * Productivity
    ~
    ~     |
Change plus defect discovery factor = 1/120
    ~
    ~     |
Cumulative Anti Regressive Work =INTEG(Anti regressive
work,0)
    ~
    ~     |
Cumulative Fielded Functionality = INTEG(Software release,0)
    ~
    ~     |
Cumulative Progressive Work  = INTEG(Implemented,0)
    ~
    ~     |
Demand obsolescense = Work Identified * 0.05
    ~ Changes/Month
    ~     |
F PROGRESSIVE FRACTION = 0.3
    ~
    ~     |
Fielded Functionality Satisfying Current Needs =
INTEG(Software release-Requirement Change flow,150)
    ~  [0,?]
    ~     |
Implemented =
IF THEN ELSE(In Progress > 0,
Progressive effort*Productivity,0)
    ~
    ~     |
In Progress = INTEG(Submision Flow-Implemented+Rejected
as needing rework,100)
    ~  [0,?]
    ~     |
IN PROGRESS TARGET  =100
    ~
    ~     |
INTEGRATION PRODUCTIVITY FACTOR = 2
    ~
    ~     |
INTEGRATION SUCCESS FACTOR = 0.95
    ~
    ~     |
NORMAL PRODUCTIVITY = 2
    ~ Modules/Person Month
    ~     |
Preparation effort =Progressive effort *PREPARATION
EFFORT MULTIPLIER
    ~
    ~     |
PREPARATION EFFORT MULTIPLIER = 0.5
    ~
    ~     |
Preparation Flow = Productivity*Preparation effort*
PREPARATION PRODUCTIVITY FACTOR
    ~
    ~     |
PREPARATION PRODUCTIVITY FACTOR = 2
    ~
    ~     |
Productivity =NORMAL PRODUCTIVITY*
(  (TEAM SIZE^0.2) - (  (1/1800) *TEAM SIZE^2)  )  *
(1-MAX(0,SYSTEM TYPE MULTIPLIER*
(Cumulative Progressive Work - Cumulative Anti Regressive
Work) ))
    ~ Modules/Person Month
    ~     |
Progressive effort = F PROGRESSIVE FRACTION*
TEAM SIZE
    ~
    ~     |
Rejected as needing rework  =
Productivity*Validation and Integration effort*
(1-INTEGRATION SUCCESS FACTOR)*
INTEGRATION PRODUCTIVITY FACTOR
    ~
    ~     |
RELEASE POLICY  ([(0,0)-(100,10)],(0,0),
(11,0),(12,1),(14,1),(15,0),
(23,0),(24,1),(26,1),(27,0),
(35,0),(36,1),(38,1),(39,0),
(47,0),(48,1),(50,1),(51,0),
(59,0),(60,1),(62,1),(63,0),
(71,0),(72,1),(74,1),(75,0),
(83,0),(84,1),(86,1),(87,0),
(95,0),(96,1),(98,1),(99,0),
(107,0),(108,1),(110,1),(111,0),
(119,0),(120,1),(122,1),(123,0),
(131,0),(132,1),(134,1),(135,0),
(143,0),(144,1),(146,1),(147,0))
    ~
    ~     |
Requirement Change flow =Fielded Functionality Satisfying
Current Needs*
Change plus defect discovery factor
    ~ Changes/Month
    ~     |
Software release =MAX(0,(Work Ready to Release/TIME
STEP)*
LOOKUP EXTRAPOLATE(RELEASE POLICY, Time))
    ~
    ~     |
Submision Flow = IF THEN ELSE((In Progress<IN
PROGRESS TARGET)
:AND:(Work Prepared for Implementation > 0),300,0)
    ~
    ~     |
Successfully integrated = IF THEN ELSE
(Work Implemented > 0,
Validation and Integration effort*Productivity*
INTEGRATION SUCCESS FACTOR * INTEGRATION
PRODUCTIVITY FACTOR,0)
    ~
    ~     |
SYSTEM TYPE MULTIPLIER =0.0005
    ~
    ~     |
TEAM SIZE = 30
    ~
    ~     |
TIME STEP = 0.125
    ~
    ~     |
Validation and Integration effort = Progressive effort *
VALIDATION AND INTEGRATION EFFORT
MULTIPLIER
    ~
    ~     |
VALIDATION AND INTEGRATION EFFORT
MULTIPLIER =  0.5
    ~
    ~     |
Work Accepted = INTEG(Acceptance Flow-Preparation
Flow,100)
    ~
    ~     |
Work Identified= INTEG(Additions and Changes Identified by
Others +Requirement Change flow-Demand obsolescense-
Acceptance Flow,600)
    ~ Changes
    ~     |
Work Implemented = INTEG(Implemented-Successfully
integrated-Rejected as needing rework,200)
    ~ Changes
    ~     |
Work Prepared for Implementation = INTEG(Preparation Flow-
Submision Flow,100)
    ~
    ~     |
Work Ready to Release =  INTEG(Successfully integrated-
Software release,0)
    ~  [0,?]
    ~     |
*************************
    .Control
*************************~
Simulation Control Paramaters
    |
FINAL TIME  = 100
    ~ Month
    ~ The final time for the simulation.
    |
INITIAL TIME  = 0
    ~ Month
    ~ The initial time for the simulation.
    |
SAVEPER  = 1
    ~ Month
    ~ The frequency with which output is stored.
    |
```