A Paradigm for the Behavioural Modelling of Software Processes using System Dynamics

M. M. Lehman J. F. Ramil G. Kahen
Dept. of Computing
Imperial College, London SW7 2AZ
tel +44 (0)20 7594 8214 fax +44 (0)20 7594 8215
{mml,ramil,gk}@doc.ic.ac.uk

ABSTRACT
Industrial software evolution processes are, in general, complex feedback systems. Recognition of this opens up opportunities to achieve sustained process improvement. The system dynamics approach was used in the FEAST projects to model behaviour of key process and product attributes, yielding foundations for a paradigm to achieve dynamic models of software evolution processes. It is based on top-down model refinement of a high level abstracted view of the process. Resulting models are relatively simple, when compared to others in the literature. Their simplicity, however, facilitates understanding, progressive refinement, subsequent validation, and industrial take-up.

The paper illustrates the paradigm with a model of a process highlighting a particular issue. The results of executing this model indicate how one may analyse and eventually optimise division of resources between progressive activity such as functional enhancement and anti regressive activity such as complexity control. The approach and model are based on empirical observations and interpretation of the evolution phenomena. When appropriately refined and validated to reflect a particular process, a model such as this can more generally be used for release planning and process improvement. The approach, together with its models, provides the basis for management technology and tools to achieve sustainable long-term evolution of complex software and systems.

Keywords
FEAST, Feedback in Software Process, Laws of Software Evolution, Management, Process Improvement, Process Models, Software Complexity, Refactoring, Resource Allocation, System Dynamics

1 INTRODUCTION

Evolution describes the phenomenon of progressive, as opposed to revolutionary, change. Evolution in the software context occurs in many areas [leh01c], including, for example, *ab initio* development, maintenance over releases and the application and related domains. Many of these areas evolve concurrently, influencing one another. Software evolution *per se* describes the process of progressive change and enhancement, with new versions or releases of the software, made available to users, as appropriate. Evolution is an intrinsic need of *E*-type software[1], arising from the need to maintain stakeholders satisfaction [leh85]. Most systems upon which businesses and organisations rely for their operations are of the type *E*, hence its importance. In this, as in previous studies [www01], we focus in the study of the *E* type.

Under systematic evolution, many systems outlive the periods for which they were originally conceived. Evolution is an expensive process. As reflected in published maintenance data, evolution is believed to consume up to 80 percent of the lifetime expenditure. If this expenditure is applied to increasing the capability and effectiveness of the system, then this may be a sound investment. To ensure that this is so requires sound planning, management and control. Nevertheless, the topic of long-term evolution management has been the Cinderella of software engineering research. What is needed, is, a sound discipline for long-term software evolution *management* [leh01b], the focus of the present paper. It presents an approach for the planning and management of long-term *software evolution* processes. The approach is exemplified by a *system dynamics* [for61,ven00] model. This model has potential for guiding decision procedures and for leading to decisions regarding resource allocation policies and related issues in software evolution processes.

The approach presented addresses three aspects of the evolution phenomena: continuing software change, continuing growth and a trend to increasing complexity. Together with earlier models [leh98,wer99,cha99,kah00], it exemplifies an approach developed during the FEAST (Feedback, Evolution And Software Technology) projects [www01] to study the behaviour of long-term evolution processes. The approach provides guidelines for building system dynamics models that reflect specific software systems and evolution domains, and thereafter for direct use as a tool in the context of their industrial processes for software evolution planning, management and process improvement.

2 CHARACTERISTICS OF THE *E*-TYPE EVOLUTION PROCESSES

E-type evolution is driven by the need to maintain stakeholder satisfaction within a changing application and usage domain. The ultimate goal of the process is to at least maintain, and generally to increase, stakeholder satisfaction in a changing operational domain. It entails adaptation of existing properties, functionality in particular, and the addition of new capability. The latter implies system functional growth over calendar time or by releases. Installation of the software changes the operational domain [leh85,ant01]. Therefore, some change requests result from the introduction of system releases into use. Others arise from user learning, familiarisation. Still others arise from market forces, human interest and ambition, advances in technology and other factors and agents exogenous to the application and system. All of resultant forces for change interact.

[1] Software being actively used to solve a problem or address an application within a real world domain [leh85]

Managing this complex situation raises the question whether *evolution processes* can be characterised so as to facilitate their management. It is a truism to affirm that behaviour of *E*-type software processes does not obey precise laws of nature as do those that govern phenomena in Physics, for example. The role of humans in all aspects of system design, implementation, and usage, and in the usage domains guarantees that. Observation and measurement of software processes at a high-level of abstraction has, however, led to positive identification of *qualitative empirical generalisations* [col64]. The latter represent elements of a characterisation. Success in behavioural process modelling will, therefore, be increased if the models are consistent with, and reflect a degree of understanding of, them.

Characteristics that have been identified by those concerned with the design and management of such processes [e.g. for61,71,80,sen90,coy96] or that may be expected to emerge in software and other complex socio-technical processes include:

- *Counter-intuitive behaviour:* Links between behaviour and its triggers are not easily identified. Effects persist long after their causes have ceased to operate. This may, for example, lead to unanticipated outcomes from implementation of management policies, if not properly identified and addressed
- *Tight coupling between process attributes*: There are likely to influence one another in unexpected ways
- *Non-linear relationships*: An additional challenge in process modelling, prediction, control

Other common characteristics of industrial software evolution processes relevant to the present discussion have been identified by a series of empirical investigations over a period of some 30 years [leh69,74,78,85,89], most recently as part of the FEAST/1 and /2 [www01] projects (1996-2001). These studies have assembled evidence supporting the suggestion that such processes tend to display similar behaviours at some level abstraction. Some of these have been encapsulated in statements, termed *laws* [leh74,78,85], to indicate that the phenomenon they reflect appear to be, at least in part, beyond the scope of control of individual process participants. That is, the behaviour implied by the laws is likely to emerge unless the laws are explicitly acknowledged and process participants act in co-ordination to compensate for their effects. Table 1 displays the current statement of the laws. The laws are numbered and presented in the order of their identification. Column one indicates the year in which the phenomenon was first recognised.

The laws have been identified across systems and processes representing a wide variety of domains. Each has been progressively refined to reflect current understanding and interpreted in terms of the observed phenomenon. Laws I, II, VI, VII and VIII are strongly supported by reasoning [leh85,01d]. With regards to the empirical evidence during the FEAST investigation, the situation is as follows: laws I and VI have been *directly* supported by the data, though without a fully satisfying distinction between relative intensity of the growth and change phenomena. Both phenomena are in general observed together. All the laws with exception of VII can be linked in varying degrees to *indirect* metric evidence. The most recent view is that the regularities suggested by laws III, IV and V are stronger over individual segments or stages of the system lifetime, hence their refinement. No empirical data has been available to us to verify law VII, though as said, this law is supported by reasoning. There are inevitably differences in the strength of the regularities across processes and over intervals of the lifecycle and their clarification requires further study. There appears, for example, to be a relation between these intervals and a recent description of stages in the software lifecycle [raj00].

The eighth law provides a formal statement of the feedback system nature of *E*-type evolution processes. It states that, except perhaps for the less mature, software evolution processes are complex feedback systems [bel71,leh74,85,94]. The reference here is to the *full* (also termed *global* [leh94]) software process. This includes the activities of all those involved, not only developers, but also their managers, support personnel, marketeers, users and so on. The other seven laws encapsulate various aspects that can be rationalised in terms of interacting feedback influences. That is, the eighth law abstracts the forces and mechanisms that underpin the other seven. The majority of the commonalties observed can be interpreted within the feedback-system view of the process.

Observation of industrial projects, their modelling and accounts of experiences provide supporting evidence of the role and impact of feedback [leh85,abd91,bro95] on the process. Evidence has been obtained from experiments with subjects in simulated environments (*flight simulators*, for example) of software process management [e.g. dra00] and other domains [sen90,kle93,ste94,lan98]. This suggests that failure to foresee and take into account feedback effects will, in many cases, explain failure to meet cost, schedule and quality targets. It also explains why software process improvement is so difficult to achieve [leh94]. On the other hand, one may expect that appropriate behavioural process models have potential an important tool towards mastering process behaviour and towards its improvement. This reinforces the suggestion that the use of behavioural process modelling techniques that focus on feedback influences such as system dynamics [for61] are promising [rio77,abd91]. If this is to be so, one needs modelling approaches that overcome inbred scepticism and facilitate industrial take up of the models and the technology they encapsulate.

In summary, there is a now a reasonable level of confidence that the laws reflect the likely long-term behaviour of *E*-type software processes. They have practical implications, not only for management of the process but also for software engineering in general [leh01c]. For example, rules and guidelines have been recently derived from the laws [leh01b]. It follows that the laws offer a context and framework within which to build behavioural process models. The latter must be models, in the mathematical sense, of a potential theory underlying the laws [leh01d].

3 SYSTEM DYNAMICS MODELLING

System dynamics [for61] techniques have been proposed and used to model real world socio-technical and other complex processes to support policy making and assessment [coy96]. From the start, the field was conceived and applied to examine the behaviour of complex processes through computer modelling and simulation tools [mea72] such as the Vensim tool [ven00] used to develop and execute the model presented in this paper. In particular, application of the approach, enables one to study process behaviour in terms of its feedback loop structure, often called the *systemic* structure. That is, system dynamics is a tool that, when properly applied, can help software managers to deal with the *systemic* properties of their decision environments. Since the focus in system dynamics is on capturing the feedback structure of the system involved, the technique was expected to be particularly appropriate to modelling evolution processes, hence its usage here.

System dynamics has previously been used to model software processes [abd91,ara93,mda96]. It should be noted, however, that, with few exceptions [ara93], the focus of interest of many of these studies appear to have been in *ab initio* development [abd91,mda96], not in long-term evolution. Moreover, they were all based on one or two step development of fine-grained models. Such an approach is likely to produce large and complex models that, though possibly of value within the context of the individual process they reflect, tend to be difficult to grasp, utilise or generalise. Thus they are not easily ported from process to process and are difficult to calibrate, interpret and exploit. This may explain, at least in part, why they have not found widespread industrial acceptance as decision-making tools. The general tendency in system dynamics to build large, complex models has yielded results that, defy intellectual mastery. To avoid this one must impose a top-down discipline to the modelling process. This insight provided the basis and rational for the approach [ram00a].

Table 1: Current statement of the laws

No.	Brief Name	Law
I 1974	Continuing Change	*E*-type systems must be continually adapted else they become progressively less satisfactory in use
II 1974	Increasing Complexity	As an *E*-type system is evolved its complexity increases unless work is done to maintain or reduce it
III 1974	Self Regulation	Evolutionary attributes of *E*-type system evolution processes tend to display a degree of statistical regularity
IV 1978	Conservation of Organisational Stability	Average work rate in an *E*-type process tends to remain constant over periods of system evolution
V 1978	Conservation of Familiarity	In general, the average incremental growth of *E*-type systems tends to remain constant or to decline
VI 1991	Continuing Growth	The functional capability of *E*-type systems must be continually increased to maintain user satisfaction over the system lifetime
VII 1996	Declining Quality	Unless rigorously adapted to take into account changes in the operational environment, the quality of *E*-type systems will appear to be declining
VIII 1996	Feedback System (Recognised 1971, formulated 1996)	*E*-type evolution processes are multi-level, multi-loop, multi-agent feedback systems

4 A PROCESS MODELLING PARADIGM

The present approach has emerged from experiences accumulated during earlier process modelling work [leh98,wer99,cha00,kah00]. Elements were already described in [leh01a]. A key ingredient is that model development follows a top-down successive refinement process [zur68,wir71] and its further elaboration in the LST development paradigm [leh84]. The idea is that, starting at a high-level of abstraction, the model is achieved by a sequence of refinements (reification) based on successive transformation and validation steps. The output of each transformation step becomes the input to the next step. The process terminates when a process model that reflects the level of granularity and precision desired is achieved in, for example, the context of policies or improvements to be assessed. In summary, the FEAST approach to behavioural process modelling includes the following activities:

- gather historical data and derive any process phenomenology
- identify *reference modes* [coy96], patterns and trends observed in real process behaviour. These are inputs for model validation
- identify specific questions to be answered by means of process modelling
- start at the highest level of abstraction possibly with a high level description of the process to be modelled
- construct an initial model that reflects only elements that are considered essential, keeping detail to the minimum necessary
- calibrate and validate model's output against real world behaviour
- iterate refinement and validation steps until the appropriate level of detail is reached

Some comments on the approach follow. A system dynamics model can reflect a system at many levels. It is important to identify an appropriate starting level of abstraction or aggregation. Subsequent refinement must lead, in a disciplined fashion, to a model that is appropriate for the purpose for which it is being constructed. In model building and refinement the recommendation is to aggregate or even exclude some of the real world detail. In particular those elements believed to be constant or of second order are initially left out. Only influences that may change significantly over system lifetime must be reflected in the model, at least in the first instance.

Elements not often varying significantly over the period studied are considered constant. They are not explicitly represented in the model. If and when they change significantly, as when a new behavioural stage emerges due, for example, to fundamental changes in the process, it must be treated as a *structural change*. The latter can be suitably (dynamically) modelled by changes during model execution in model structure and parameters, as appropriate.

This procedure may appear to be self-evident. It is, however, our experiences that it is of value to describe such procedure. This is particularly so since the focus in process modelling research has, in the past, been more on formalisms than on methodology.

5 ILLUSTRATING THE PARADIGM

5.1 Rationale Underlying the Model

Increasing complexity is, in general, primarily due to, and evidenced by, greater inter-element connectivity. As functionality is added by applying change upon change, functional *interdependence* increases, interfaces become more complex (sic) and the system grows in size. The consequent higher level of reciprocal interdependency will, itself, lead to more control mechanisms. *Information hiding* [par72], and other good evolutionary practices [leh01b] may make a contribution to the control of complexity growth, though mostly through prevention rather than correction. However, even under the best of conditions and processes, evolving *E*-type systems *without* a degree of intrinsic system complexity growth does not appear to be possible. This follows from the fact that the application implemented by the evolving system is increasing in complexity. That is, the many causes of inherent complexity growth during system evolution include complexity growth of the application and of the operational domain. Developments in software architecture, on the other hand, particularly architectures that are component based, may reduce such complexity growth or even mitigate it. We hypothesise, however, that even in component-intensive processes the constraints implied by increasing complexity will emerge at a higher level of aggregation or abstraction [leh00].

Given that these consequences must accompany software evolution, a minimum level of complexity management and control activity are essential to sustain the rate of system evolution at the required or desired level over a lifetime that is commensurate with the investment in it.

5.2 Anti-regressive Activity

In a 1967 study the leading economist W J Baumol applied a categorisation of *progressive* and *anti-progressive* to distinguish between activities in the work place that had or had not a potential for growth in their productivity [bau67]. The paper was drawn to the attention to one of the present authors (mml) who noted that in the context of the software process there was a vital third category which he termed *anti-regressive* [leh74,85]. This was distinguished by its having zero or even negative productivity in that it absorbed production resources without adding to the immediate, marketable, value of the product. What such activity contributed was durability, a potential for the product to retain its value in the future. Lehman recognised that this, in the form of complexity control and reduction to compensate for the inevitable complexity growth of evolving software, was a vital concept in software evolution management. Another example was the preparation and maintenance of technical documentation that would enable continued evolution of the software long after those who created it had become unavailable. In other words, anti-regressive activity prevents or circumvents system decline due to non-material factors, it prevents system decline. Thus, the new term anti-regressive was used to refer to work effort intended to compensate for *aging* effects [par94] or code decay [eic01]. Such work consumes effort without any *immediate, perceived* stakeholder return in system value as reflected, for example, by system functional power or performance. Instead, it facilitates continued evolution, enabling it to be achieved more easily with less effort.

In planning and executing sustainable evolution a portion of the work must be devoted to complexity control otherwise the software is likely to become a legacy system. There is anecdotal evidence that in some processes as much as 30 percent of the work is devoted to complexity control, though it may not be explicitly identified as such. Thus, in general, the following question arises: what is the optimum level of complexity control activity in the long-term evolution context? There have been already attempts to address this question [rio77,leh85]. The modelling work presented here follows in that tradition. The model presented in the following sections presents an attempt to address the above question, based on current modelling technology [ven00] and showing how the modelling approach described above may be used to identify complexity control mechanisms that operate by balancing resource allocation between progressive and anti-regressive activities. Though focusing here on the specific issue of complexity control [leh74] the approach described is, in principle, applicable to all aspects of compensation for software aging effects [par94], an issue that under the term *refactoring* has recently has received wide attention [fow99].

5.3 The Model

The diagrams and terminology of system dynamics provide one approach to modelling the dynamic behaviour of long-term software evolution processes. The model presented here takes the form of a *stock and flow diagram* [ven00]. It is a graph that consists of two connected networks: the *stock and flow* network, and the *information* network. Levels (or stocks) are represented by variables within the rectangles. The double-line arrows represent flow variables (or rates). The remainder of the model, represented by single line arrows and variables constitutes the information network. The single lines indicate that the variable close to the arrow's origin has an immediate influence on the variable being pointed to. Further details on the semantics of the diagrams are given in [coy96,ven00].

The model will be presented in a series of four figures that present increasing detail. This exemplifies the process of successive refinement that has been advocated. This approach clearly is of major benefit to the development of understanding of the model. An initial top-level view of is provided by figure 1. This represents the view of the software evolution process as a service process, by showing the arrival and implementation or rejection of work requests, their validation, and delivery of the product to the field. It is visualised as a process that addresses a continuing flow of work. To ensure consistency of the model with industrial practice, the model structure was discussed with FEAST industrial collaborators at various stages of its development and reflects their comments where appropriate.

Figure 2 presents a first refinement of the model, making provision for the holding back of the output of the validation step and for the authorisation of rework. Figure 3 refines the model further to include the assignment of resources to progressive work. Finally, figure 4 incorporates additional elements such as the variables and mechanisms involved in the split of effort between progressive and anti-regressive activities.

The model is built around a simple mechanism already proposed in [rio77]. It is of striking simplicity. It enables one to deal with the problem of managing complexity without actually having to define a complexity metric [zus90], an issue that is still an open issue. Its further clarification requires, *inter alia*, a formal definition of complexity as a function of measurable program attributes. However, the present model is built upon the observation that a degree of anti-regressive activity may control the complexity as perceived by the developers/evolvers of the system so as to minimise its impact. This implicitly defines complexity indirectly by means of the observed global effects. The mechanism relies on the fact that for each unit of progressive work accomplished, a minimal amount of anti-regressive work is required, otherwise an anti-regressive deficit begins to accumulate. As the anti regressive neglected work accumulates over time, its impact on productivity begins to be noticeable. Only its restoration to an adequate level can reverse that growing trend and seek to restore productivity. A tool, such as this model is required to determine what is adequate.

Resource allocation is represented by two variables: *Progressive Effort* and *Anti-regressive Effort.* A key policy parameter is indicated on the right-hand side of Figure 3 by the fraction of the personnel dedicated to progressive activities, as represented in Vensim formalism [ven00] as:

Progressive Effort =
*= (1- “PROGRESSIVE ANTI REGRESSIVE RATIO")*TEAM SIZE*

Space limitations prevent inclusion of the other equations constituting the model. The full set is available from the authors on request.

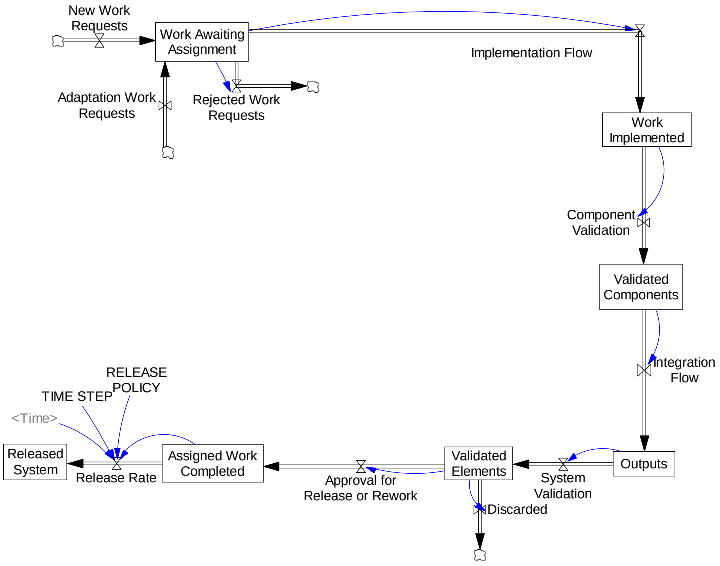

Figure 1: The starting view

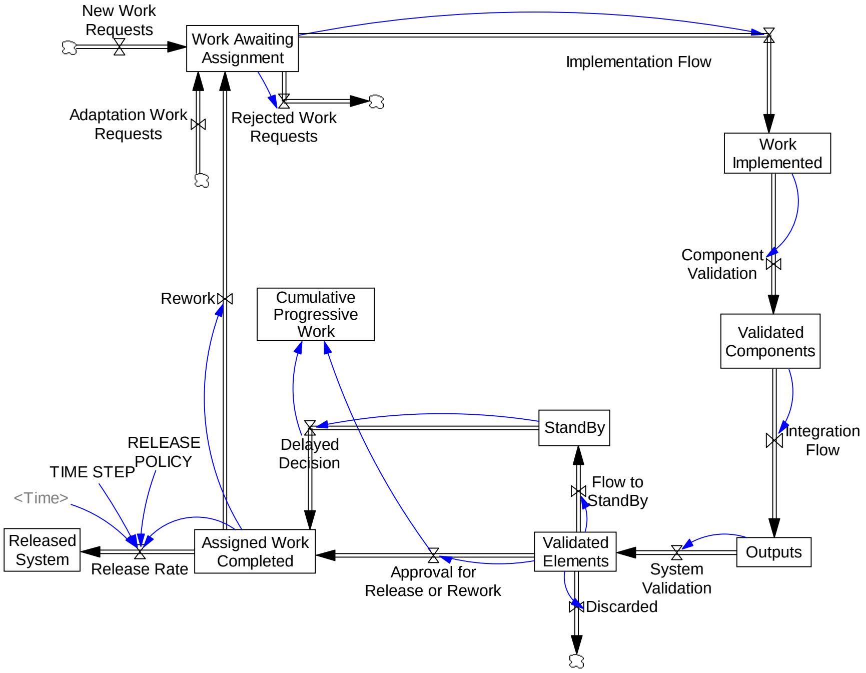

Figure 2: The model after the first refinement: some part of output held, rejected or recycled

5.4 Further Model Refinement

Once the portion of resources to be dedicated to progressive and anti-regressive activities respectively has been selected, the model assigns a fixed portion of progressive effort to *Implementation, Component Validation, Component Integration* and *System Validation*. This is to reflect the fact that, on average, none of these activities can be neglected. Otherwise the neglected activity will produce a bottleneck. It will be clear to the reader that all these assignments represent simplifications to permit initial, high level, validation of the model against real industrial processes. Further refinements can be applied, preferably one at the time, to reflect and investigate the factors arbitrarily assigned as above, based on such model output behaviour that can be explained in terms of current understanding of the processes, or of changed understanding as a result of model interpretation.

5.5 Model Calibration

As already stressed, an essential part of the approach being described is that the model be validated [for80] at the appropriate level of detail after each refinement. One possible way of achieving this is by calibrating the model, comparing predictive model output with actual behaviour.

Validation and calibration may start by exposing the model to available data so that confidence in the model increases progressively. Such increases will, generally, be accompanied by growing understanding of the process being modelled.

As a first step in the calibration process, parameters must be set to be consistent with the process being modelled. This implies measurement of real world attributes indicated by the model. Once those values are obtained, one sets model parameters to reflect real values. In practice, some of these may not be readily available, as for example when, as in the present case, one is modelling long term behaviour. It may, therefore, be necessary to, identify ranges of parameters that produce specific behaviours. In doing so, the model builders start to identify which parameters are critical in determining specific behaviours. Then one proceeds to check with process experts and/or by using documentary sources the possible values in the real process, thereby, building confidence in the model. Of course, this only yields a partial calibration. A full calibration requires that all model parameters reflect actual values. On the other hand, validation requires that the model is shown to *predict* real world behaviour not observed during model building or calibration and remains accurate over time. Hence, calibration and validation must be continuing activities.

These are, of course, important activities that are required to build confidence in the model and the emphasis here is in the *approach* and not in the particular model being presented. It is, nevertheless, interesting to note that a relatively simple model (by comparison with system dynamic models as presented by others that may involve tens or even hundreds of variables [e.g. abd91]) such as the one presented in figures 1-4 can so closely approximate real world patterns of behaviour. Figure 5 shows how closely the model reproduces the growth trend of one software system over 176 months of its lifetime. This system is an operational support system. Its evolution trends, as measured by its growth and cumulative modules handled, are broadly similar to those of other software systems studied in FEAST.

As illustrated by figure 5 calibration, to the extent performed, shows that the model is able to replicate actual trends despite the fact that, in general, not all model parameters are known. By fixing the known parameters and exploring the effects of the others, one establishes the sensitivity of the model to the various parameters. At this stage of model development it may, of course, not be possible to determine whether such sensitivity, or its absence, is a property of the process or of the model. In some cases, however, reasoning may suggest a solution to this fundamental question. For example, in the present model, it was evident that the value of parameters representing the flows feeding to *Work Awaiting Assignment* were relatively unimportant with respect to the rest of the model as long as there was "enough work waiting". This was, in fact, accepted as an appropriate property since it implements common experience in real world evolution processes that the work waiting queue tends never to be empty. This may, in fact, be a general characteristic of the type introduced in section 2.

In some cases, non availability of direct measures forces one to use attribute surrogates. For example, to calibrate the model to this system, the level of effort applied (*STAFFING POLICY*) was assumed to be roughly proportional to the workrate metric, *modules handled* [leh85] that, in a later study [ram00b], was found to be correlated with estimates of the effort applied. Other parameters were found to have no visible impact on growth trend within a range of values. This suggested that the mechanism to which they related had no major impact on growth trend at the present level of detail; that it was not a candidate for calibration adjustments (and hence for process control) in the current setting. The inflexion point at around month 96 was modelled by a step change in the value of the variable *Impact of Anti-regressive Deficit*. To date we have been unable to identify the real world cause of the inflexion point, or its source in a change in the process or the implementation domain. It may reflect a switch of process stages in the sense of other researchers [raj00,ant01].

Model parameters whose value could not be readily ascertained were set to values that minimised the difference between the actual growth trend and model outputs. Of course, the full validity of values obtained in this way to advance, for example, understanding of the process and its model representation, depends on confirmation of its validity by, for example, successful behavioural prediction. With the conclusion of the FEAST/2 project we were unable to pursue the next stage of model refinement by determining actual values of the data related to the parameters and recalibrating. However, the model as presented here suffices to exemplify the approach.

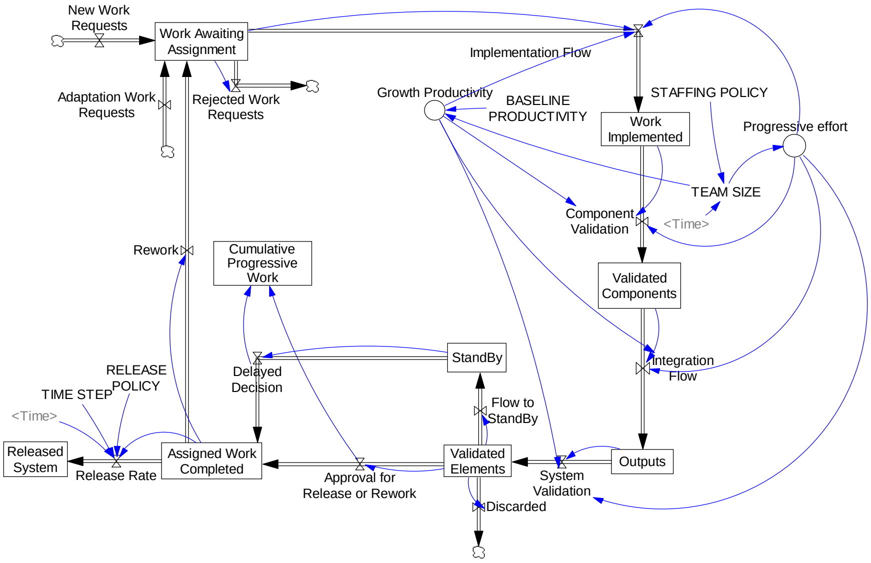

Figure 3: The model after the second refinement: resources for new function

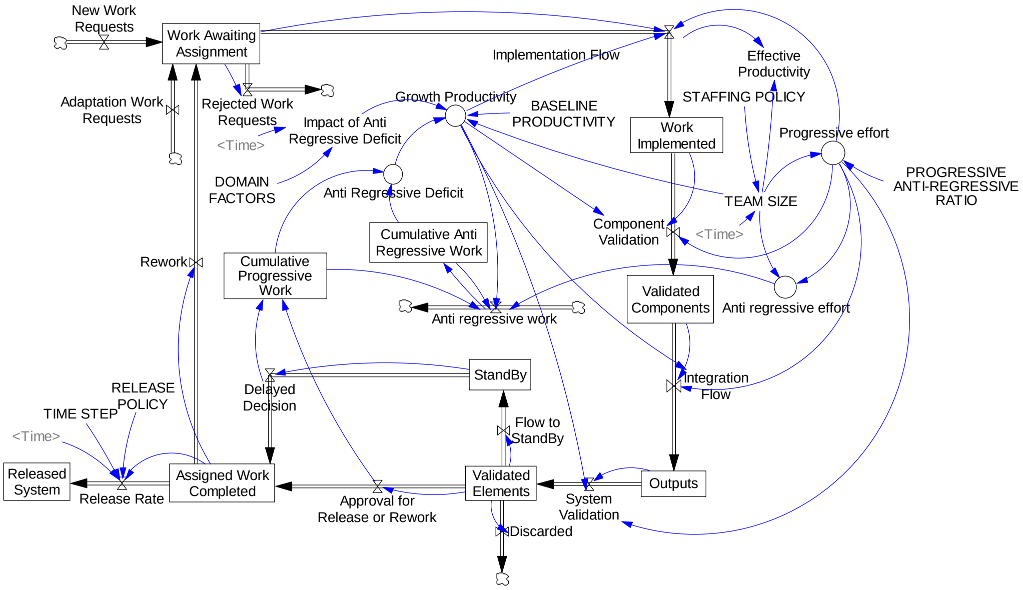

Figure 4: The model after the third refinement: resource allocation for complexity control

5.6 Virtual Experiments

Given a model in which one has confidence that it reflects real world behaviour, one can usefully perform virtual experiments. The term virtual is used here to highlight the fact that these experiments are performed *with the model*, not in the real world software organisation and process. There are, however, classes of questions, such as policy evaluation for long-term evolution, that, by their very nature, can only be pursued by means of models. One would require to have, for example, two evolution processes running more or less in parallel for which one is willing and able to adopt conflicting policies and whose impact one is able to monitor, measure and evaluate. But in general one must rely on virtual experiments with models that, to some degree, represent the observed phenomena. The results of such experiments that follow must then, of course be handled intelligently.

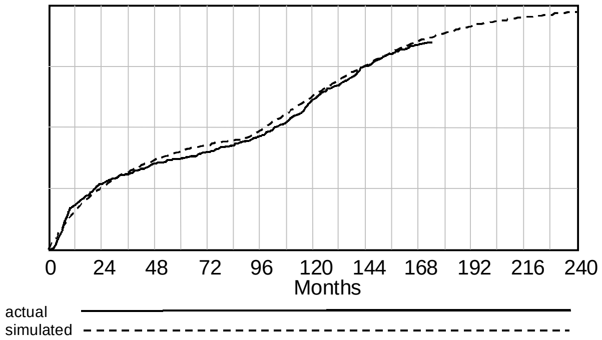

Figure 5: Simulated model output vs actual growth trend (a collaborator system)

The example of a virtual experiment to be outlined is that of the long-term consequences of different levels of anti-regressive activity on the system growth rate. The result of one such experiment is shown in Figure 6. It suggests that allocation of a portion of resources to anti-regressive work results in significant extension of the potential life span of the system.

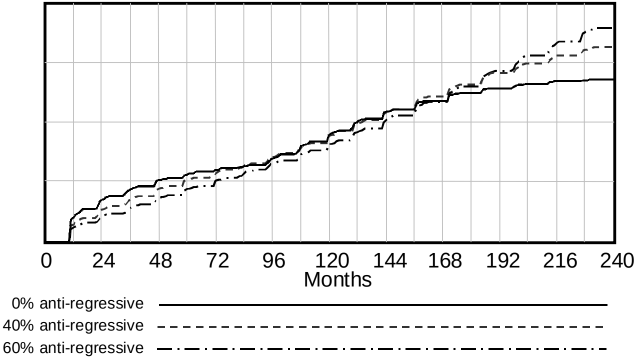

Figure 6: Simulated model output for 3 values of anti regressive work, expressed in percentage of total resources. Several varying parameters make it difficult to identify the effects

The trends reflected in Figure 6 include, however, temporal variations in effort applied and an inflexion point. To simplify interpretation, one can investigate the impact of parameter changes, one parameter at the time. This is illustrated in Figures 7 and 8 where, to isolate the inflexion point issue from this analysis, the model was fitted to the first growth segment only. Execution of the resultant model permits a clearer visualisation of the effect of different anti-regressive policies. The simulated growth trends for various values of anti regressive effort, given these experimental changes outlined above, are presented in figure 7.

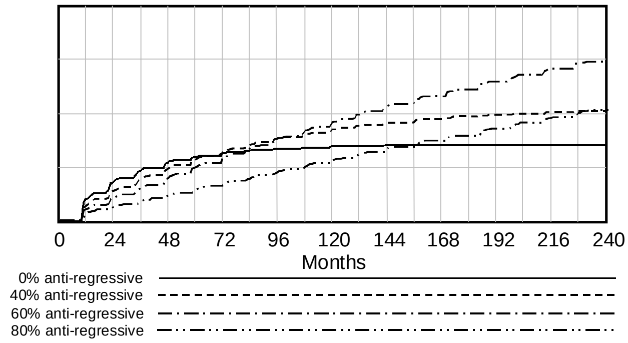

Figure 7: Growth trends under different policies

In Figure 7 one observes several cross-over points amongst the trajectories. Those that reflect a low level of anti-regressive work provide the highest *initial* accomplishment rates. This suggests that a sound strategy would start by selecting trajectories with low anti-regressive activity, thus maximising the initial growth rate.

In figure 8 one observes the impact of different levels of anti regressive work on productivity. The overall conclusion from both figures is that, even with a constant a level of anti regressive work there is one which maximises long-term growth capability (in this case approx. 60 percent). Anti regressive work in excess of such level, represents resource wastage.

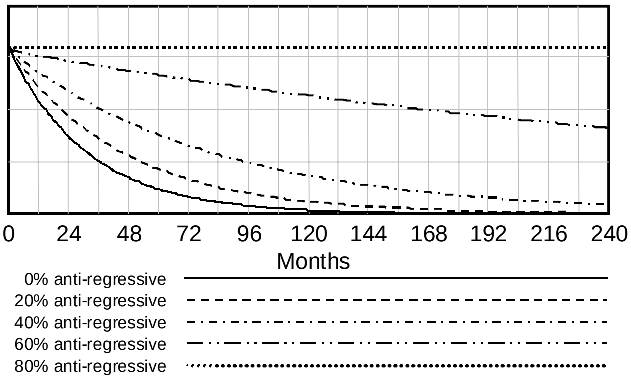

Figure 8: Growth productivity under different policies

A simple interpretation of these diagrams suggests that one can switch trajectories at cross-over points by changing the progressive anti-regressive ratio so as to always sit on that trajectory with the highest growth rate. However, experiments that cannot be accommodated within the limitations of this paper have shown that this is not sufficient and system re-structuring, will be also required. This reflects a beautiful example of how behaviour of a feedback system is often counterintuitive. In any event, whether restructuring occurs or not, figure 7 suggests that as the system ages one may seek to maintain system growth rate – or, equivalently, the effort required to achieve the desired rate of evolution, and, hence, productivity - by increasing the level of anti-regressive activity.

The optimum level of anti-regressive work is likely to vary from process to process and from stage to stage in the life cycle of a given system. To best manage this process and to allocate resources to maximise benefit requires an approach, models and tools such as those we have explored, provisionally tested and presented here. More detailed policies and mechanisms such as the inclusion of a management feedback control loop that changes the degree of anti-regressive activity over time in response to some simulated circumstances, should only require minor model modification for their exploration.

In summary, one may conclude that:

- The intended emphasis of this paper is on the modelling approach. The model presented provides a tool, which, when calibrated to a particular process, can be used to evaluate the impact of different policies and managerial decisions concerning effort allocation
- It is also, in general, a prototype of the models required to determine the impact of policies and managerial decisions
- Differences are expected from process to process, and from stage and stage, within the same process, in specific policies and the evolution dynamics as a whole. To remain useful the modelling process must parallel and reflect the evolution process and system evolution
- The model provides a generic view that can be adapted by structural and equational changes to reflect specific evolution processes. One can pursue further refinement through extension and addition of detail

6 RELATED WORK

The principal focus in process modelling has been in modelling formalisms and languages [ost87], in presenting and analysing specific models [ispw], not in the methods by which such models are derived. In contrast to data collection methodology, that has received some attention [e.g. bas84], process modelling methodology has not been the focus of much research.

Additionally, the vast majority of software process simulation models have had the individual *ab initio* project context as their focus. Only a few exceptions, exemplified by the model in Aranda *et al* [ara93] and the FEAST white box process modelling work [leh98,wer99,cha99,kah00], has the full evolution process [leh94] and long-term evolution as their focus. A related characteristic of the modelling work presented here, also recognised by others [e.g. rui00], is the representation of the process at a high-level of abstraction. This is sharp contrast to the wider effort that has chosen to investigate the process at a low level of abstraction, leading to complex models that are difficult to comprehend, absorb, calibrate, utilise and reuse.

7 THE WAY FORWARD

For wide application, the model presented will need to be refined to accommodate, for example, allocation of work to a wider variety of activities than just anti-regressive and progressive work. Several classifications of maintenance and evolution work have been proposed over the years. They may serve this purpose [cha00]. One could further address the split between the other categories of effort, though in principle it appears that anti-regressive effort provides a major element to sustain process effectiveness over the entire system life cycle.

Extension to more general paradigms and customisation to specific instances is likely to raise issues not considered here. These would likely include measures of stakeholder satisfaction and the impact of this on the dynamics. A particular factor cannot be achieved without a satisfactory definition that is probably domain dependent. This question requires further exploration and may involve the inclusion of system *value* [boe00]. Another aspect, previously considered [cha99] and that requires further study and development, is the role of domain and technology changes as drivers of software evolution. Furthermore one will have to compare the role of discrete and continuous activities, for example, the role and impact of field trial events [leh98,wer99] as opposed to the continuing validation activity assumed in the present model. Finally, in this highlighting of aspects for future investigation, we mention the consequences of different release policies [woo79]. All these exemplify issues that could be studied by means of the approach presented.

8 FINAL REMARKS

Behavioural process modelling have been pursued in the software engineering community for many years. The approach indicates that, even when the processes are executed and managed by humans whose individual decision-making behaviour is essentially unpredictable, the process models are useful, a source of insight and provide a rational for decision making. The local process will, at each instant in time, be the result of and reflect local decision making in the context of locally perceived circumstances. Process modelling at an aggregated, high level of abstraction, reflects process behaviour at that level. Thus, the approach and the model presented here can assist managers to recognise and control the various influences on long-term behaviour. By taking them into account, they may then direct effort to specific activities that otherwise would have been ignored or neglected. They may also be able to predict and *control* key evolutionary attributes and reduce the likelihood of counterintuitive behaviour. Society relies increasingly on software, as demonstrated by the essential role of software in businesses and organisations. Management of long-term evolution processes is becoming an ever more critical issue. Such processes are very complex and dynamic. There is long way to go before a full understanding is reached. We foresee that systematic arrangement of the concepts and principles that are now being established may provide software process technology with the conceptual framework it so sadly lacks [leh01d].

9 ACKNOWLEDGEMENTS

Many thanks are due to the FEAST industrial collaborators for provision of data, interpretations and for sharing with us their process insight. Financial support from the UK EPSRC, most recently via grant number GR/M44101 (FEAST/2) (1999-2001), is gratefully acknowledged.

10 REFERENCES[2]

[abd91] Abdel-Hamid T. & Madnick S., “Software Project Dynamics - An Integrated Approach“, Prentice-Hall, Englewood Cliffs, NJ., 1991

[ant01] Anton A & Potts C, “Functional Paleontology: System Evolution as the User Sees It”, ICSE 23, Toronto, 12-19 May 2001, pp. 421-430

[ara93] Aranda R *et al*., "Quality Microworlds: Modeling the Impact of Quality Initiatives over the Software Product Life Cycle", Am. Programmer, 6(5), 1993, pp. 52-61

[bas84] Basili V.R. and Weiss D.M., “A Methodology for Collecting Valid Software Engineering Data”, IEEE Trans. on Softw. Engineering, v. 10, n. 6, 1984, pp. 728 - 737

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

[boe00] Boehm B.W. & Sullivan K.J., “Software Economics: A Roadmap“, in Finkelstein A (ed.), The Future of Software Engineering, ICSE 22, Limerick, Ireland, 4-11th June 2000, pp. 321 - 343

[bro95] Brooks F. P., “The Mythical Man-Month”, 20th Anniversary Edition, Reading, Massachusetts: Addison Wesley Longman, 1995

[cha00] Chapin N. *et al*, "Types of Software Maintenance and Evolution", ICSM 2000, 11-13 Oct. 2000, San Jose, CA

[cha99] Chatters B. W., Lehman M. M., Ramil J.F. & Wernick P., “Modelling a Software Evolution Process“, ProSim'99, Softw. Process Modelling and Simulation Workshop, Silver Falls, Oregon, 28-30 Jun. 1999, also as Modelling a Long Term Software Evolution Process in J. of Softw. Proc.: Improvement and Practice, Vol. 5, iss. 2/3, July 2000, pps. 95 - 102

[cho81] Chong-Hok-Yuen C.K.S., "A Phenomenology of Program Maintenance and Evolution", PhD thesis, Imperial College, Dept. Computing, 1981, 302 pp

[2] An * indicates that the reference has been reprinted in [leh85].

[col64] Coleman J.S., "Introduction to Mathematical Sociology", The Free Press Of Glencoe, Collier-Macmillan Limited, London, 1964, 554 pps.

[coy96] Coyle, R.G., "System Dynamics Modelling - A Practical Approach", Chapman & Hall London, 1996, 413 p

[dra00] Drappa A. & Ludewig J. "Simulation in Software Engineering Training", Proc, ICSE 2000, June 4-11th, Limerick, Ireland, pp. 199 - 208

[eic01] Eick S.G., Graves T.L., Karr A.F., Marron J.S. and Mockus A., "Does Code Decay? Assessing the Evidence from Change Management Data", IEEE Trans. on Softw. Eng., v. 27, n. 1, Jan. 2001, pp. 1 - 12

[for61] Forrester, J. W., "Industrial Dynamics", MIT Press, Cambridge, Mass., 1961

[for71] id., "Counterintuitive Behaviour of Social Systems", Technology Review, v. 73, 1971, pp. 52-67

[for80] Forrester, J. W. & Senge, P., "Tests for Building Confidence in System Dynamics Models", In System Dynamics, Legasto A. A. Jr., et al. (eds.) TIMS Studies in the Management Sciences, Vol. 14. North Holland, NY, 1980, pp. 209 - 228

[fow99] Fowler M., "Refactoring: Improving the Design of Existing Code", Addison-Wesley Longman, NY, 1999

[ispw] Intl. Softw. Process Workshop, Proceedings 1991-1996, IEEE Computer Society Press, Los Alamitos, CA

[kah01] Kahen G., Lehman M. M., Ramil J. F. & Wernick P. D., "Modelling of Software Evolution Processes for Policy Investigation: Approach and Example", to appear in J. of Systems and Software, 2001, a revised version of paper presented at ProSim 2000 Workshop, July 12 - 14, 2000, Imp. Col., London UK

[kit82] Kitchenham, B.A., "System Evolution Dynamics of VME/B", ICL Tech. J., May 1982, pp 42 - 57

[kle93] Kleinmuntz, D. N., "Information Processing and Misperceptions of the Implications of Feedback in Dynamic Decision Making", System Dynamics Review, v. 9, 1993, pp. 223-237.

[lan98] Langley, P. A., "Using Cognitive Feedback to Improve Performance and Accelerate Learning in a Simulated Oil Industry", WP-0027, LBS, UK, 1998

[leh69]* Lehman, M. M., "The Programming Process", IBM Res. Rep. RC 2722, IBM Res. Centre, Yorktown Heights, NY 10594, Sept. 1969

[leh74]* Lehman, M. M. "Programs, Cities, Students, Limits to Growth?", Inaugural Lecture, 1974,, 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.

[leh78]* Lehman, M. M., "Laws of Program Evolution _ Rules and Tools for Programming Management", Proc. Infotech State of the Art Conf., Why Software Projects Fail?, 1978, pp. 11/1-11/25

[leh84] Lehman, M. M., Stenning V. and Turski W., "Another Look at Software Design Methodology", ACM Softw. Engineering Notes, v. 9, n. 2, 1984, pp. 38-53

[leh85] Lehman, M. M & Belady, L. A., "Program Evolution - Processes of Software Change", Acad. Press, 1985

[leh89] Lehman, M. M., "Uncertainty in Computer Application and its control through the Engineering of Software", Journal of Software Maintenance, Research and Practice, Vol. 1, 1989, pp. 3-27.

[leh94] Lehman, M. M., "Feedback in the Software Evolution Process", Keynote Address, CSR Eleventh Annual Workshop on Software Evolution: Models and Metrics. Dublin, 7-9th Sept. 1994, Workshop Proc., Information and Softw. Tech., sp. is. on Softw. Maint., v. 38., n. 11, 1996, Elsevier, 1996, pp 681 – 686

[leh98] Lehman M. M. & Wernick P., "System Dynamics Models of Software Evolution Processes", Proc. IWPSE-98, 20-21 Apr. 1998, Kyoto, Japan, pp. 6-10

[leh00] Lehman MM & Ramil JF, "Software Evolution in the Age of Component Based Software Engineering", IEE Proc. Softw., sp. issue on Component Based Software Engineering, v. 147, n. 6, Dec. 2000, pp. 249 - 255

[leh01a] Lehman M. M., Ramil J. F. and Kahen G., "Experiences with Behavioural Process Modelling in FEAST, and some of its Practical Implications", EWSPT 8, 21 June 2001, Haus Bommerholz, Witten, Germany, LNCS 2077, Springer, Berlin, 2000, pp. 47-62

[leh01b] Lehman, M.M. & Ramil, J.F., "Rules and Tools for Software Evolution Planning and Management", to appear in special issue on Software Management, Annals of Software Engineering, vol. 11, 2001, revised version of a FEAST 2000 contribution and as DoC Research Report 2000/14 Nov. 2000

[leh01c] Lehman, M. M. and Ramil, J. F., "Software Evolution", inv. lect., Pre-prints IWPSE 2001, Vienna, Sept. 10-11, to be publ. in Proc, IWPSE 2001, IEEE Press

[leh01d] *id.*, "An Approach to a Theory of Software Evolution", *loc. cit.*

[mda96] Madachy, R., "System Dynamics Modeling of an Inspection-based Process", Proc. ICSE 96, Berlin, Germany, March 25 - 29, 1996, pp 376 - 386

[mea72] Meadows, D.H. *et al*, "Limits to Growth", Signet, 1972

[ost87] Osterweil L, "Software Processes Are Software Too", Proc. ICSE 9, 1987, pp 2 - 12

[par72] Parnas, D.L., "On the Criteria to be Used in Decomposing Systems into Modules", CACM, v. 15, n. 12, pp. 1053-8

[par94] Parnas, D. L., "Software Aging", Proc. 16th ICSE, May 16-21, Sorrento, Italy, 1994, pp. 279-287

[pau93] Paulk, M.C. et al, "Capability Maturity Model", Version 1.1, IEEE Softw., v. 10, n. 4, 1993, 18 – 27

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

[ram00a] Ramil, J.F., Lehman M.M. and Kahen G., "The FEAST Approach to Quantitative Process Modelling of Software Evolution Processes", Proc. PROFES 2000, June 20 - 22, 2000, in Frank Bomarius and Markku Oivo (eds.) LNCS 1840, Springer, 2000, pp. 311 - 325

[ram00b] Ramil, J.F. and Lehman M.M., "Metrics of Software Evolution as Effort Predictors - A Case Study", Proc. ICSM 2000, October 11-14, 2000, San Jose, CA, pp. 163 – 172

[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

[rui00] Ruiz, M. & Ramos I., "A Dynamic Estimation Model for the Early Stages of a Software Project", ProSim 2000, Workshop on Software Process Simulation and

Modelling, Imperial College, London, 12-14 July, 2000, http://www.prosim.org <as of July 2000>

[sen90] Senge, P.M., "The Fifth Discipline - The Art & Practice of The Learning Organisation", Currency- Doubleday, NY, 423 pp., 1990

[ste94] Sterman, J., "Learning in and about Complex Systems", System Dynamics Review, Vol. 10, 1994, pp. 291-330.

[ven00] Vensim - Ventana Simulation Environment, Reference Manual, Version 4, Belmont, MA, 2000

[wer98] Wernick P. & Lehman M. M., "Software Process White Box Modelling for FEAST/1", ProSim '98 Workshop, Silver Falls, OR, 23 Jun. 1998. As a rev. version in J. of Systems and Software, v. 46, n. 2/3, 15 April 1999

[wer00] Wernick P, "Identifying and Justifying Metrics for Software Evolution Investigations Using the Goal-Question Metric Method", FEAST 2000, July 10-12, 2000, Imperial College, London, http://www.doc.ic.ac.uk/~mml/f2000 <as of July 2000>

[wir71] Wirth, N., "Program Development by Stepwise Refinement", CACM, v.14, n.4, Apr., 1971, pp. 221-227

[woo79] *Woodside, C.M., "A Mathematical Model for the Evolution of Software", ICST CCD Res. Rep 79/55, Apr. 1979. Also in J Sys and Software, Vol 1, No 4, Oct 1980, pp 337 – 345

[www01] FEAST "*F*eedback, *E*volution And *S*oftware *T*echnology", FEAST/2 Project Web Site, 2001 http://www.doc.ic.ac.uk/~mml/feast/

[zur68] Zurcher, F.W. & Randell, B., Iterative Multi-Level Modeling - A Methodology for Computer System Design, Proc. IFIP Congress 1968, Edinburgh, Aug 5 - 10, pp D-138 – 142

[zus90] Zuse H, Software Complexity Measures and Models, de Gruyter, NY, 1990