An Approach to System Dynamics Modelling of Aspects of the Global Software Process

Running Title: System Dynamics of Global Software Process Aspects

G Kahen MM Lehman JF Ramil
Department of Computing
Imperial College of Science, Technology and Medicine
180 Queen's Gate
London SW7 2BZ, UK
tel +44 20 7594 8214 fax +44 20 7594 8215
{gk, mml, ramil}@doc.ic.ac.uk

P Wernick
Department of Computer Science
University of Hertfordshire
College Lane
Hatfield AL10 9AB, UK
tel +44 1707 286323 fax +44 1707 284303
p.d.wernick@herts.ac.uk

Address for correspondence and proofs

Professor Manny Lehman
Department of Computing
Imperial College of Science, Technology and Medicine
180 Queen's Gate
London SW7 2BZ
United Kingdom

Abstract

This paper describes one of the latest in a series of system dynamics models developed during the FEAST (Feedback, Evolution And Software Technology) investigation into software evolution processes. The intention of early models was to simulate real-world processes in order to increase understanding of such processes. The work resulted in a number of lessons learnt, in particular, with regard to the application of system dynamics to the simulation of key attributes of long-term software evolution. The work reported here combines elements of previous work and extends them by describing an approach to investigate the consequences on long-term evolution, of decisions made by the managers of these processes. The approach is illustrated by discussion of a model developed using the *Vensim* tool. This model of the impact on product and global process attributes of decisions regarding the fraction of work applied to progressive and to anti-regressive activities such as complexity control, for instance, exemplifies the results of the FEAST investigation.

Keywords: anti-regressive activity, decision making, *E*-type systems, evolution, FEAST, feedback, global software process, laws of software evolution, management, planning, progressive activity, simulation, software process modelling, system dynamics, white-box modelling

1 Introduction

Observations emanating from studies of attributes of the sequences of releases of OS/360-70 (Lehman, 1969) and other software systems, initiated during the late 1960s and continued during the 1970s and 1980s, led to the identification of the *software evolution phenomenon* (Lehman, 1974; Lehman and Belady, 1985). Evolution is here understood as the continuing fixing, adaptation and enhancement of software systems, version after version, release after release within changing application domain and stakeholder needs. Results of the earlier studies of the software evolution phenomenon included, *inter alia*, the laws of software evolution (Lehman, 1974; 1978; 1980; 1991; 1996), a classification of software into types *S*, *P* and *E* (Lehman, 1980), a principle of software uncertainty (Lehman, 1989; 1990), and, more recently, the FEAST hypothesis (Lehman, 1994).

Almost since its inception, the investigation of the evolution phenomenon has included modelling of process dynamics. This is exemplified by three models of evolutionary system growth developed in the 1970s (Belady and Lehman 1975; Riordan, 1977;Woodside, 1979). These models, inspired by observed growth patterns of evolving software and by reasoning about the process, explored the impact on long-term functional growth of pursuing different evolution policies. However, it was only in the early 1990s that modelling of software process dynamics gained wider interest, though these studies (Abdel-Hamid and Madnick, 1991) concentrated on *ab initio* projects. Modelling work addressing long-term software life cycle issues followed (e.g. Aranda et al., 1993; Wernick and Lehman, 1998; Williford and Chang, 1998; McCray and Clark, 1999; Chatters et al., 1999).

With the collaboration of ICL, Logica and Matra BAe Dynamics the FEAST/1 project (1996–1998) (Feast, 2001) was able to substantiate, refine and extend the earlier results. This was made possible by analysis of data provided by these companies on the evolution of their respective systems, VME Kernel, the FW Banking Transaction system and a defence system. Data on two real-time Lucent Technologies systems also became available for analysis during this time. In FEAST/1, *system dynamics* (SD) (Forrester 1961; Coyle, 1996) and the *Vensim* tool (Vensim, 1995) were used to build models of two of

the industrial software processes being investigated (Wernick and Lehman, 1998; Catters et al., 1999). The continuing model building work has produced models which are relatively simple when compared with other SD models reported in the literature (e.g. McCray and Clark, 1999).

One of the objectives of FEAST/1 was to achieve a better *understanding* of the long-term dynamics of the *global* software process (Lehman, 1994). Such a process includes the activities of *all* those involved: developers, users, marketing and support personnel and their managers, etc. It encompasses, but extends beyond, the immediate technical software process . The on-going FEAST/2 project (1999–2001) (Feast, 2001), with BT Labs as an additional collaborator, has continued the investigation. It has produced amongst its results a set of management and planning guidelines (Lehman, 2000a), which are derived from analysis and interpretation of the evolution phenomenology gathered over the years (Lehman, 1974; Lehman and Belady 1985; Feast, 2001). Other results of the project included a set of lessons learnt in the application of quantitative models to the study of software processes (Ramil et al., 2000).

The present paper describes and exemplifies an approach to the application of SD models to the investigation of evolution policies. This work aims at contributing to the development of techniques for strategic management, planning and control of long-term software-product evolution. The need for modelling work addressing these issues within a long-term view has been highlighted in a summary of work in software process modelling (Kellner et. al., 1999). The approach proposed in the present paper combines elements presents in early models (Belady and Lehman, 1975; Riordan, 1977; Woodside, 1979), insights gained in FEAST/1 (Wernick and Lehman, 1998; Chatters et al., 1999) and those obtained in FEAST/2 (Feast, 2001).

The present paper refers exclusively to *E*-type software system evolution, that, the evolution of systems regularly used to solve a problem in a real-world domain. The majority of the systems upon which businesses and organisations depend for their operation are of this type, hence its importance. Such systems display the intrinsic property that their functional capability must be continually adapted and enhanced, as long as they are in regular use (Lehman and Belady, 1985). Otherwise they deteriorate in effectiveness. The main criterion by which *E*-type software is judged is stakeholder satisfaction.

The results presented in this paper are restricted in their applicability to traditional software evolution paradigms. Whether, and to what degree, they are relevant in the context of emerging paradigms such as reuse-intensive processes, component-intensive and COTS-based software[1] remains to be investigated. Evolution of open source software is also, here, excluded from the discussion.

2 Process Modelling in FEAST

The FEAST projects have focused on the long-term behaviour of product and global process attributes, such as system size and rate of work. In so doing they have pursued two classes, termed *black-box* and *white-box,* respectively, of software process modelling. The former focuses on (mathematical) models that reflect externally observed process behaviour. This modelling activity looks primarily at patterns and structure *in the metric data*. White box models, on the other hand, reflect the operations of elements within the actual process. The two approaches are complementary and may be used in parallel, sequentially and/or iteratively as part of quantitative modelling support for evolution software process management and improvement (Ramil et al., 2000). The present paper focuses on white-box modelling. References to the black-box modelling work may be found on the FEAST web-site (Feast, 2001).

The following list characterises the FEAST white-box modelling work. To date it has, *inter alia*:

- sought to *understand* the phenomenon of software evolution and to *identify* significant *feedback* loops that drive evolutionary behaviour. This has included the development of models with a degree of explanatory power, not only such as focus on prediction of process attributes
- focused on modelling of long-term attributes with time units of years or, alternatively, release sequence numbers. Release sequence numbers have been used as *pseudo time* indicators (Cox and Lewis, 1966), because the release point in time represents an instance in time where the system, its attributes and the code that implements them are clearly determined by the state of the product delivered. At other times, the artefacts manipulated by the process are in state of flux.
- placed emphasis on models with a small number of variables. Models can then be refined by means of

[1] Some of the issues relating to the long-term evolution of component-intensive software have been discussed in Lehman and Ramil, 1998.

a top-down approach (Zurcher and Randell, 1967) to achieve behaviour closer to that observed in the specific processes of interest and subsequently validate, calibrate and use the refined models as decision-aids

- modelled the evolution processes and product attributes of processes studied using metrics such as module counts, which are considered to have a higher level of functional or semantic integrity than lines-of-code or other similar alternatives and hence, more meaningful in the context of long-term evolution studies.

3 The Approach

This section presents the basis for a system dynamics (SD) modelling approach to investigate and evaluate *E*-type system evolution policies. One starts by seeking to identify the major classes of influences that should be modelled to permit such evaluation. The following list identifies the elements considered by the present authors in their work. The ordering of the items is of no particular significance. Many others items could, of course, have been selected. The choice in any particular study will reflect the influences in a given application and evolution domain believed to be significant in that study, the concerns to be explored.

- CONSTRAINTS – the make-up of the relevant process together with the related organisational and managerial bounds and limits. Such constraints might include, for example, the maximum number of personnel and the time frame available for the evolution of a particular system. Others constraints could emerge from contractual agreements and from the economic environment and reward practices within which the evolution takes place. Empirical underlying observations such as those expressed in Brook's law (Brooks, 1995, p. 274), and the laws of software evolution (Lehman 1974; Lehman and Belady 1985; Feast, 2001) can also be considered constraints
- DYNAMICS OF DEMAND – characterisation of the demand for evolution work (e.g. rate of incoming work requests from users and other stakeholders). Demand may be described by attributes such as, for example, volatility of existing requirements, rate of arrival of change requests, changes in

type and/or size of the user constituency that impact on, for example, user response to changes in the software, step changes in demand due to market, legislation and so on

- DYNAMICS OF RESOURCES – representing the size, skills, knowledge and experience, familiarity with evolving application, productivity, effectiveness and/or other relevant characteristics of the team responsible for software system evolution. These would relate to activities such as work identification, planning and preparation, analysis and design, implementation, validation and verification, support, sales, etc
- DYNAMICS OF TECHNOLOGY – similar to the dynamics of the demand, but reflecting technological (software, hardware, process technology, design methodology and tools, etc.) domains and the work demand therefrom generated
- TECHNICAL EVOLUTION PROCESS – attributes of the immediate technical process steps such as preparation and planning, implementation, verification & validation, and rework rates that are relevant in an investigation of policies
- PAYOFF FUNCTION(S) – the relationship(s) that are sought to be optimised over the long-term evolution process. This makes it possible to *formalise* in the model the objective(s) that managers are seeking or will seek to optimise, for example, the functional power of a system, its impact on stakeholder satisfaction, the contribution of the system to the businesses it supports, etc;
- POLICIES – such as those implicit in the operation of open or feedback-loop mechanisms, that are likely to include human decisors who seek optimisation of payoff functions and/or the stabilisation[2] of the behaviour of the attributes of the product and/or the process over long-term evolution process. Such attributes include, for example, productivity, system growth, portion of system *handled* per release (Lehman and Belady, 1985), evolution effort, including release interval and content, system complexity and so on etc
- STAKEHOLDER SATISFACTION – a set of attributes that reflect the degree of achievement of

[2] Stabilisation here may include not only the achievement of constant values but also in some cases of constant change rates.

stakeholders' needs and desires by the evolving system. This may (or may not) explicitly form part of the payoff function, depending on whether its optimisation is actively sought. An alternative formulation is that stakeholders' satisfaction needs to meet or exceed some minimum threshold

- STAGES OF SOFTWARE EVOLUTON – Bennett and Rajlich have recently discussed the role of stages in the long-term evolution of software products (Bennett and Rajlich, 2000; Rajlich and Bennett, 2000). In one stage, for example, the emphasis may be on implementation of new function. In another stage the emphasis may be on adaptation and enhancement of existing functionality. A further stage may focus on essential fixing work to keep the system in operation. When one considers the evolution situation in and across stages one must expect to find differences in the models applied to investigate evolution policies. Thus, SD models may have to account for different stages for the models to remain useful.
- STRUCTURAL CHANGES - Product and global process changes that are significant enough may impact the dynamics of the global process. These may be for example, changes in the user base, organisational changes or radically different methods and tools from those used previously. They may also be exemplified by major addition or change in requirements, major system reengineering, addition of a major subsystem with new functionality, changes in programming language. In a SD model, these changes may be appropriately accounted for by step changes in model parameters. Other cases may require more fundamental changes to the model.

The model builders will find that some of these elements may be modelled, based, for example, on expert knowledge, relevant historical data and on the application of forecasting techniques.

The inclusion in the model of elements that may be particularly uncertain, such as those related to the dynamics of demand, may be achieved by modelling them as explicit assumptions to reflect, for example, different scenarios. The above list identifies some of the elements present, explicitly or implicitly, in some of the earlier models (Belady and Lehman, 1975; Riordan, 1977; Woodside, 1979, Wernick and Lehman, 1998; Chatters et al., 1999) and in the model presented in the next section as an example of the approach.

4 The Model

It is generally agreed that the term software evolution encompasses activities that address functional fixing, adaptation, and enhancement of a software system over its lifetime. (Swanson, 1976). Within that broad description several classifications of maintenance and evolution activity, as summarised in a recent paper (Chapin et al., 2001), have been proposed over the years. One such classification, proposed by Lehman in 1974, is believed to have a significant implication in the context of long-term evolution management. It relates to the successive superposition of changes on individual modules of the evolving software. Such sequences are likely to lead to increasing complexity of the software and also give rise to other aging effects (Lehman 1974; Lehman and Belady, 1985; Parnas, 1994). If not adequately compensated for, these lead, to a decrease in productivity of the evolution process (Swanson, 1999). Following Baumol's classification of work effort into *progressive* and *anti-progressive* types (Baumol, 1967), Lehman proposed a further category, *anti-regressive* (Lehman 1974; Lehman 1985). Baumol's term *progressive* was applied to evolution activities that enhance system functionality by modification of or addition to the code and/or the documentation. The term *anti-regressive* was used to refer to work effort intended to compensate for the software aging effects. Such work consumes effort without any immediate visible stakeholder return as reflected, for example, by system functional power or performance. Instead, it facilitates further evolution, reducing the effort required and enabling it to be achieved more quickly and more reliably. than would otherwise be required. But anti-regressive work in excess of some threshold may represent an investment of resources whose beneficial contribution to future evolvability is of less value than the contribution that resource could have made to progressive system evolution. The achievement of an adequate balance between the two categories is crucial to enable further evolution in a cost-effective manner.

The approach and the SD model presented here are intended to explore the impact on product and global process attributes of policy decisions regarding the balance between progressive and anti-regressive work. The objective is to provide a model that, once refined and calibrated to represent a real world process and its environment, can be used as a tool to explore the impact of different policies and

support decision making in this regard.

Figure 1 shows a SD model that, in the main, addresses the problem of long-term system growth and explores the consequences of different levels of anti-regressive work. This involves the top right-hand portion of the model. The model also includes the view of the global process as a closed loop. It reflects the assumption, that delivery of functionality to the field is a source of future work requests as visible on the lower left corner of the model. Finally, the equations embedded in the two elements '*Progressive'* and '*Anti Regressive Work Productivity'* force a non-linear relationship between productivity and team size. This is inspired by Brook's observation that as the team size increases, productivity losses in communication and similar tasks will grow by a factor that is proportional to the square of the team size (Brooks, 1995). The details of all these aspects of the model are given in the equations provided in the Appendix.

This model inherits elements from earlier models (Belady and Lehman, 1975; Riordan, 1977; Woodside, 1979, Wernick and Lehman, 1998; Chatters et al., 1999). From the models of the 70s it inherits the view that an imbalance between cumulative progressive and cumulative anti-regressive work leads to deterioration in productivity. This allows one to consider the effect of growing complexity without having to select a software complexity measure. From the models of the 90s cited, the model inherits the view of the global process as a closed loop.

The model is constructed in *Vensim* (Vensim, 1995) and is fully executable. Following the rationale underlying the simplifications made in the earlier FEAST SD models, the excluded elements are currently assumed either to be constant, and not to impact dynamic behaviour, or of not significantly affecting the particular policy being investigated. The structure of the model and the relations it incorporates have been discussed with some of our industrial collaborators. The model considers the software evolution process as a process of arrival, implementation, delivery and generation of *change requests*[3]. The term *change* is used here in a wide sense to encompass both new and changed requirements and the work required to

[3] The model embeds the assumption that one can aggregate all the evolution activity in a sequence of homogenous change request units. One

implement them in the evolving software[4]. This includes verification or validation activities and deployment of the new version of the software in the operational domain.

In principle, the model could equally well be based onto other measures of evolution effort. As already implied the actual selection of a particular measure must depend on data availability and other considerations. One also requires measures of *work achieved, growth and/or change in functional power,* for example and measures of change. The latter could reflect numbers of requirements units changed (e.g. paragraphs in a requirements specification document), number of function points added and changed, number of implemented addition and modification requests, counts of modules modified, counts of modules *handled* and number of changed lines of code.

Definitions of model variables in the Appendix refer, in general, to *changes*, without reference to any particular measure. The relative merits of the candidate measures remain to be fully explored. Note that different measures may display the attribute at different levels of granularity and from different perspectives. It may, therefore, be beneficial to retrieve data and to develop a model family that can operate to operate on data reflecting several measures. It will then be of interest to determine whether the several measures reflect analogous general patterns and trends, sometimes called *reference modes* (Coyle 1996) in SD terminology. If significant behavioural differences are observed then these must be investigated and clarified.

The Appendix includes descriptions of all the variables in the model. We refer here only to the two that are directly related to the example that follows. These are *'Anti-Regressive Work Policy'* and *'Threshold Productivity'*. In Fig. 1, *'Anti-Regressive Work Policy'* represents the fraction of resources assigned to such work and is a fraction that may take any value between 0 and 1. 0 means that no resource is assigned to anti-regressive work, and 1, implying that *all* available resources are applied to that activity. The model simulates a management feedback control mechanism by starting the anti-regressive work if and when a decrease in productivity occurs with respect to a given *'Threshold Productivity'*. Model output

---

could refine the model to reflect at a higher level of granularity the existence of different types of work.

(Fig. 2), with *'Anti-Regressive Work Policy'* set to 0, reproduces closely the growth trend of one of the systems studied in FEAST/2 over 180 months of its lifetime to late 1999.

The software system from which the growth trend was derived is a large operational support system. Its evolution trends (growth, cumulative modules handled) are broadly similar to those of other software systems studied in FEAST. The actual growth trend in Fig. 2 (solid line) was extracted by analysing change-log records of source code. For this purpose several *Perl* scripts were developed. The script identifies the date of creation of each module, accumulates the number of modules created per month thereby enabling determination of the growth trend. Model parameters were adjusted so that the difference between the actual growth trend and model output was minimum. Although one must not presume that the similarity of the curves in Figure 2 indicates that the model is an adequate abstraction of the real process (this would have required, amongst other steps, the checking the values of model parameters against real data derived from the process being modelled. Unfortunately, such data were not available at the time of writing). It is, nevertheless, interesting to note that a model that is relatively simple by comparison with other published models that tend to involve hundreds of variables) can so closely approximate real world patterns. But model validation requires matching of model and real world mechanisms for it to be accepted as significant and meaningful. Model parameters have been set to values that are tentative in terms of our understanding of the phenomenon being simulated. This does not, however, constitute a valid calibration against a real world process nor can such a calibration be undertaken without preparatory work that relates the model mechanisms to real process mechanisms or procedures.

5 Model Outputs

As an example of the use of the model for policy evaluation, Figs. 3 and 4 show the significant impact of policy decisions on the predicted behaviour of productivity under simulated conditions over the next 90 months of system evolution, that is for a period beyond the point in time for which actual system growth

[4] In this paper we do not make a distinction between changes to existing code and the addition of new code. All is subsumed by the term *change*.

data was available.

In particular, Fig. 4 shows that the model predicts that 30 percent of resources assigned to anti-regressive work results in significant extension of the potential system life span. This is only an example of a possible policy; one that would be refined in successive steps. More detailed policies, such as changing the degree of anti-regressive activity over time in response to some simulated circumstances, can be require only minor modification of the model for their exploration.

6 Discussion

Observation of industrial processes suggests that key attributes of the product and process do not tend to be actively and consistently managed over the entire product life cycle. Management decisions tend to deal primarily with relatively short-term issues and details. In a complex feedback-based process, local decision-making is likely to lead to sub-optimal global behaviour over the entire life cycle (Lehman, 1969, 1994). In a time of increasing dependence on computer-based systems by businesses and organisations at large, it appears that there is a lack of decision models to support decisions related to long-term system evolution issues. The modelling approach and the model described in the previous sections is offered as a contribution towards that end. In general, the proposed approach may be used to support decisions related to, for example:

- what is the functional growth that an evolving system is likely to achieve over a period of time based on an evolution policy?
- is the software process likely to be able to cope with the work demand?
- would it appear to be appropriate to replace an ageing system (and/or process) by a new one? If yes, when?[5]

Answers to the above questions can be pursued from different perspectives and considering some of the many domains (e.g. application, market, technology, process) and the dimensions (e.g. stakeholder satisfaction, value, risk, cost) involved.

In the FEAST black-box modelling work the present authors have sought answer to the above questions by: retrieving data that reflect process and product attributes, detecting trends and extrapolating them using models such as the *inverse square model* of system growth (Turski, 1996; Feast, 2001). The *inverse square model* has exhibited a remarkable predictive power of growth trends over one or two individual segments, interpreted as stages, with accuracy within 10 percent of the actual size values over releases. However, such a black-box model does not account for the potential effect of either future structural changes in the global process or future active management of the growth trend. Nor do such models identify the mechanisms or phenomena to which the observed behaviour could be attributed or by changes to which t observed behaviour could be changed, improved in some sense for example. For this, white-box models such as the one described in the previous section may be useful.

Active, conscious management of growth trends and other important attributes (e.g. maintainability (Swanson, 1999) may be desirable, in particular, in some organisations that are heavily dependent on computers and software, either as a product to be merchandised or as a vital part of their internal operation. As shown in the previous section, white-box modelling techniques can be used to investigate the consequences of different evolution policies and, hence, achieve, in some sense, optimal behaviour. Note that by considering a policy not as open-loop but as a feedback control, such as the one in the example shown, a degree of robustness is added against inaccuracies in the SD model.

The parameters of the model presented here have been selected so that model output reflects a real-world growth trend. This is, in fact, considered to be the first step in what then can become a calibration based on real-world data of the rest of model parameters, relying, when possible, on historial data and/or expert knowledge. Model validation must follow by, for example, assessing predictive accuracy of the model with data sets ‘not seen’ during model calibration, tracking a ‘live’ process and examining how closely the model's output follows real-world behaviour. One must document all the assumptions made during calibration and its validation. Such validation of the model and of all relevant model parameters

[5] The model proposed in this paper can be useful in connection with the replacement decision procedures discussed in Lehman et al., 2000.

may usually trigger a sequence of model modifications and refinements until the empirical validity of the model is established on a firmer basis. The latter is essential before attempting to use the model as a decision support tool for a real evolution process. In this sense, the main contribution of this paper is in proposing an approach, and indicating with a model exemplar, *the scope and a level of abstraction* for such modelling activity.

7 Conclusions of the Modelling Exercise

This section describes the differences between this model and those developed during FEAST/1 and reported previously. It sets out the conclusions which have so far been drawn from the process of building this new model.

Differences from earlier models include:

- the explicit modelling of changes in the software system, the creation of its demand and how that demand is met, which was abstracted in earlier FEAST studies
- the splitting of effort applied during software evolution into progressive and anti-regressive activities, and the simulation of the effect of the ratio between these on changes made to the software product
- the integration of a *policy* into the model, in this case a policy which determines the amount of anti-regressive activity based on the current productivity of developers
- reflection of non-linearities in productivity

Conclusions from the earlier FEAST SD model-building work reinforced by that reported here are that:

- simulation studies such as this exemplify an application of the behavioural invariants encapsulated in the Laws of Software Evolution. Conversely, the model's success in replicating real-world behaviour, provides further support for the Laws. To increase confidence in the latter observation requires, however, that the model presented here must be fully calibrated and validated
- it is possible to simulate long-term trends in the size of real-world software products using models of this type

- the approach adopted in the development of earlier FEAST models, including features such as an emphasis on simple models, and the modelling of the software production process at a high level of abstraction, supports more detailed model-building using an step-wise incremental refinement approach (Zurcher and Randell, 1967). This approach is now being used to concentrate on specific areas of the global software process, adding more detail as the need for this is indicated from a desire for greater accuracy in the simulation outputs
- in this more complex model as in earlier simpler FEAST SD models, abstraction of areas not of current interest does not prevent the generation of broadly accurate simulation outputs.

In addition, the development and successful calibration of this new model against general trends in a real-world software evolution process which embodies structures intended to simulate both the effect of the increasing complexity of software systems suggested by the 2nd law of software evolution (Increasing Complexity) (Lehman 1974, Lehman and Belady, 1985) on the ability to modify that system and the possible effects of a policy to reduce or counteract this complexity, provides additional specific support for this law.

8 Future Work

This model forms a potential starting point for the identification of inputs and building blocks of a more detailed generic SD model of software evolution processes than has previously been possible. Adding detail to the model implies progressive identification of exogenous and endogenous influences on the behaviour of evolution processes. It forms the second step (after previous modelling exercises in FEAST) in building a theory of software evolution (Lehman, 2000b;Lehman and Ramil, 2000). Work on the development of such theory will benefit from the availability of a set of models that will enable validation and future refinement of the theory.

Steps to be taken in further refinement and use of the model presented here include the following:

- employing simulations based on adjusting model policy variables to improve decision-making in the management of evolution for a real-world software system;

- extending the model to simulate different software evolution domains, including component-intensive and COTS based software processes (Lehman and Ramil, 1998);
- exploring the possibility of building a generic model for different software application areas, development organisations, outside environments, and so on;
- extending the modelling work to form a set of building blocks, enabling the instantiation of different areas of the model and the integration of these after calibration into a common model for each application; and
- integrating it with black-box models in an iterative modelling procedure for improvement of evolution processes as suggested in (Ramil et al., 2000).

9 Final Remark

The summary of concepts relevant to the determination of policies for managing *E*-type system evolution, and their exemplification by means of a SD model, as illustrated in this paper contributes towards the achievement of systematic strategic planning, management and control of long-term software evolution. Much, however, remains to be done if quantitative models are to be successfully and widely used in industrial software evolution processes.

Acknowledgements

Grateful thanks are due to our industrial collaborators and to Professors Dewayne Perry and Wlad Turski, Senior Visiting Fellows to FEAST/2, for their technical comments on early versions of the modelling work presented here. Grateful thanks are also due to Ms Siew F Lim for her help in checking the references and to Miss Celia Gould for her help in proof-reading an earlier version. Any remaining errors are responsibility of the present authors. Financial support from the UK EPSRC, grant number GR/M44101 (FEAST/2 Project), is gratefully acknowledged.

References

Abdel-Hamid, T. K. and Madnick, S. E., *Software Project Dynamics - An Integrated Approach*, Prentice-Hall, Englewood Cliffs,

New Jersey, 1991, p. 264.

Aranda, R. R., Fiddaman, T. and Oliva, R., Quality Microworlds: Modeling the Impact of Quality Initiatives over the Software Product Life cycle, *American Programmer*, Vol. 6, No. 5, 52-61 (May 1993).

Baumol, W. J., Macro-Economics of Unbalanced Growth - The Anatomy of Urban Cities, *Am. Econ. Review*, 415-426 (June 1967).

Belady, L. and Lehman, M. M., The Evolution Dynamics of Large Programs, *IBM Res. Rep. RC5615*, T J Watson Res. Centre, Yorktown Heights, NY 13598, p. 45 (September 1975).

Bennett, K. H. and Rajlich, V. T., Software Maintenance and Evolution: a Roadmap, in *The Future of Software Engineering* (A. Finkelstein ed.), 22nd. ICSE, 4-11 June 2000, Limerick, Ireland, *ACM* Order Nr. 592000-1, 75-87.

Brooks, F. P., *The Mythical Man-Month*, 20th Aniv. Edition, Addison-Wesley, Reading, MA, 1995, p. 322. First edition appeared in 1975.

Carnap, R., *An Introduction to the Philosophy of Science*, Gardner M (ed.), Dover, Toronto, 1995, 300 pps

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 June 99. Also as: Modelling a Long-Term Software Evolution Process, *Software Process - Improvement and Practice*, vol. 5, iss. 2/3, 95-102 (July 2000).

Chapin, N., Hale, J. E., Khan, K. M., Ramil, J. F. and Tan, W. G., Types of Software Evolution and Maintenance, to appear in *Journal of Software Evolution and Software Maintenance*, to appear, *J. Softw. Maint. Evol: Res. Pract.* (2001).

Cox, D. R. and Lewis, P. A .W., The Statistical Analysis of Series of Events, Methuen, London, 1966.

Coyle, R. G., *System Dynamics Modelling - A Practical Approach*, Chapman & Hall, London, 1996, p. 413.

FEAST, Feedback, Evolution And Software Technology, 1996-2001, Projects Web Site, http://www.doc.ic.ac.uk/~mml/feast.

Forrester, J. W., *Industrial Dynamics*, MIT Press, Cambridge, Mass., 1961.

Kellner M. I., Madachy, R. J. and Raffo, D. M., Software Process Simulation Modelling: Why? What? How?, *Journal of Systems and Software*, Vol. 46, No. 2/3, 91 -106 (April 1999).

Lehman, M. M., The Programming Process, *IBM Research Report RC 2722*, IBM Research Center, Yorktown Heights, NY, (September 1969). Reprinted in Lehman and Belady, 1985.

Lehman, M. M., Programs, Cities, Students, Limits to Growth?, Inaugural Lecture, in Imperial College of Science and Technology Inaugural Lecture Series, Vol. 9, 211-229 (1970, 1974). Also in Programming Methodology, (D. Gries. ed.*), Springer Verlag*, 42-62 (1978). Reprinted in Lehman and Belady, 1985.

Lehman, M. M., Laws of Program Evolution—Rules and Tools for Programming Management, *Proceedings of the Infotech State*

of the Art Conference, Why Software Projects Fail, 11/1-11/25 (April 1978). Reprinted in Lehman and Belady, 1985.

Lehman, M. M., On Understanding Laws, Evolution, and Conservation in the Large Program Life Cycle, *Journal of Systems and Software*, Vol. 1, No. 3, 1980, pp. 213-221. Reprinted in Lehman and Belady, 1985.

Lehman, M. M. and Belady, L. A., *Program Evolution - Processes of Software Change*, Academic Press, 1985.

Lehman, M. M., Uncertainty in Computer Application and its Control through the Engineering of Software, *J. of Software Maintenance, Research and Practice*, vol. 1, 1, 3-27 (September 1989).

Lehman, M. M., Uncertainty in Computer Application, Technical Letter, *CACM*, vol. 33, no. 5, 584 (May 1990).

Lehman, M.M., Software Engineering, the Software Process and Their Support, *IEE Softw. Eng. J.*, Spec. Iss. on Software Environments and Factories, Sept. 1991, 6:5, pp. 243 - 258

Lehman, M. M., Feedback in the Software Evolution Process, Keynote Address, CSR Eleventh Annual Workshop on Software Evolution: Models and Metrics. Dublin, 7-9 Sept. 1994, Workshop Proc., Information and Software Technology, sp. is. *on Software Maint.*, v. 38, n. 11, Elsevier, 681-686 (1996).

Lehman, M. M., *Laws of Software Evolution Revisited*, Proceedings of EWSPT'96, Nancy, October 1996, LNCS 1149, Springer Verlag, 1997, pp. 108-124

Lehman, M. M. and Ramil, J. F., Implications of Laws of Software Evolution on Continuing Successful Use of COTS Software, DoC Tech. Rep. 98/8, Imperial College, London, Jun. 1998. A revised version to appear as Software Evolution in the Age of Component Based Software Engineering, *IEE Proceedings Software*, 2001.

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

Lehman, M. M., *TheSE - Approach to a Theory of Software Evolution*, Project Proposal, DoC, Imperial College, Dec. 2000

Lehman, M. M., and Ramil, J. F., *Towards a Theory of Software Evolution - And Its Practical Impact*, invited talk, Pre-prints of ISPSE 2000, Intl. Symposium on the Principles of Software, Evolution, Kanazawa, Japan, Nov 1-2, 2000, 1 - 9

Lehman, M. M., Ramil, J. F. and Kahen, G., Replacement Decisions for E-type Software - Some Elements, ICSE 2000 2nd Workshop on Economics-Driven Software Engineering Research, Limerick, Ireland, 6 Jun. 2000

McCray, G. E. and Clark, T. D., Using System Dynamics to Anticipate the Organisational Impacts of Outsourcing, *System Dynamics Review*, Vol. 15, No. 4, 345-373 (Winter 1999).

Parnas DL, Software Aging, Proc. 16th ICSE, May 16-21, 1994, Sorrento, Italy, pp 279-287

Ramil, J. F., Lehman, M. M. and Kahen, G., The FEAST Approach to Quantitative Process Modelling of Software Evolution Processes, Proc. PROFES'2000 2nd International Conference on Product Focused Software Process Improvement,

Oulu, Finland, 20 - 22 Jun. 2000, in Frank Bomarius and Markku Oivo (eds.) LNCS 1840, *Springer Verlag*, Berlin, 311-325 (2000).

Rajlich, V.T. and Bennett, K.H., A Staged Model for the Software Life Cycle, *Computer*, July 2000, pp. 66 - 71

Riordan, J. S., 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

Swanson, E. B., The Dimensions of Maintenance, Proc. 2nd. Intl. Conf. on Softw. Eng., pp. 492 - 497, 1976

Swanson, E. B., *IS Maintainability: Should It Reduce the Maintenance Effort?*, SIGCPR' 99 New Orleans LA, USA, pp. 164 - 173

Turski, W. M., Reference Model for Smooth Growth of Software Systems, *IEEE Trans. on Software Engineering*, 22 (8), 599–600 (1996).

Vensim - Ventana Simulation Environment, Reference Manual, Version 1.62, 1995

Wernick, P. D. and Lehman, M. M., Software Process White Box Modelling for FEAST/1, ProSim '98 Workshop, Silver Falls, OR, 23 June 1998. Also in *Journal of Systems and Software*, Vol. 46, No. 2/3 (April 1999), 193 -202

Williford, J. and Chang, A., Modeling the FedEx IT Division: a System Dynamics Approach to Strategic IT Planning, ProSim '98 Workshop, Silver Falls, OR, 23 June 1998. Also in *Journal of Systems and Software*, Vol. 46, No. 2/3 (April 1999), 203 -212

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, and as Chapter 16 in Lehman MM and Belady LA, *Program Evolution - Processes of Software Change*, Academic Press, 1985, p. 538.

Zurcher, F. W. and Randell, B., Iterative Multi-Level Modeling - A Methodology for Computer System Design, IBM Res. Div. Rep. RC-1938, Nov. 1967. Also in *Proc. IFIP Congress*, Edinburgh, D-138-142 (5-10 August 1968).

Figures belonging to main part of the paper

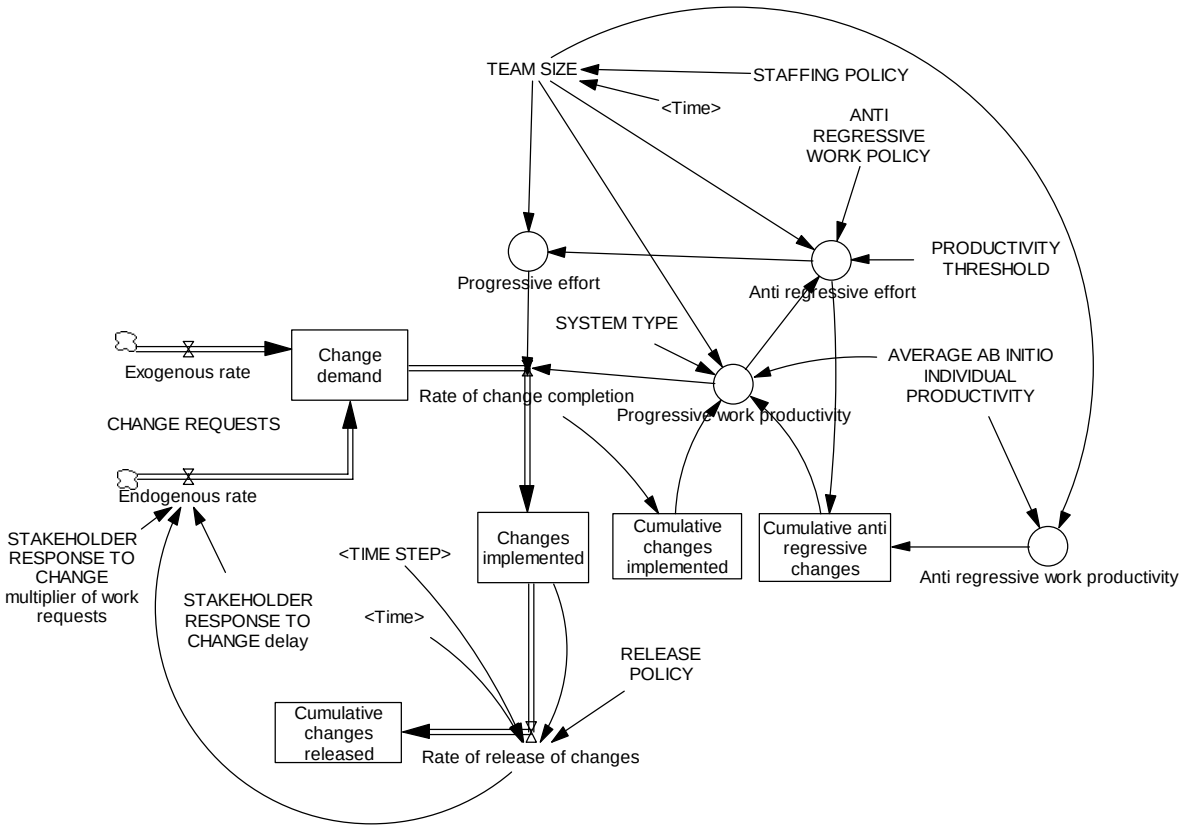

Figure 1 - System dynamics model to study aspects of the global software process

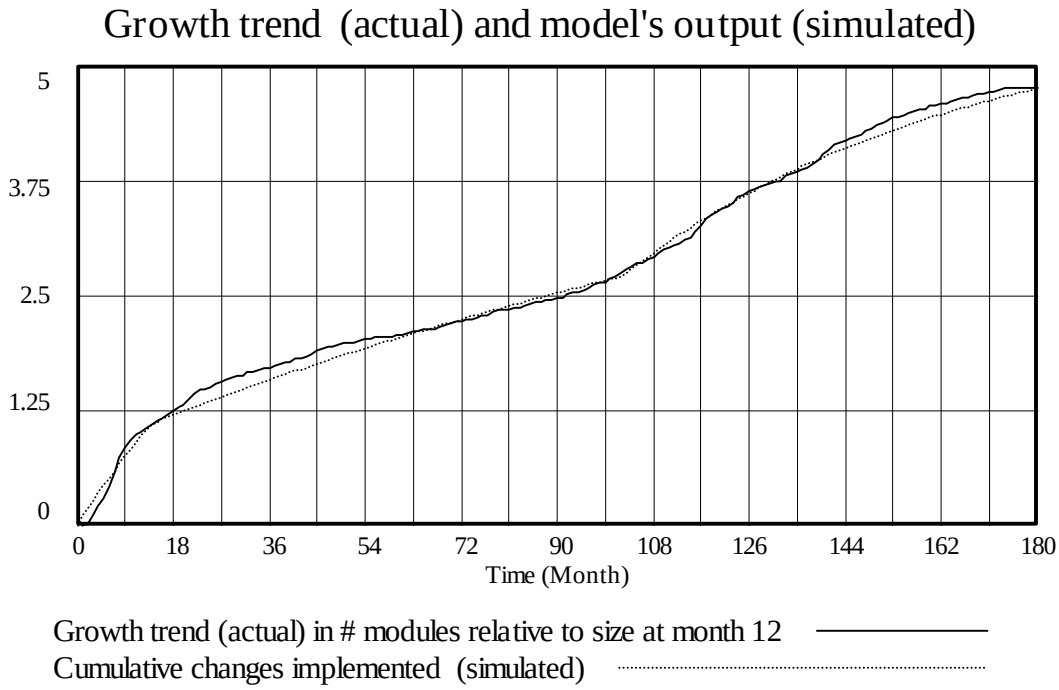

Figure 2 - Growth trend in modules and model simulated output for one of the systems studied in FEAST/2

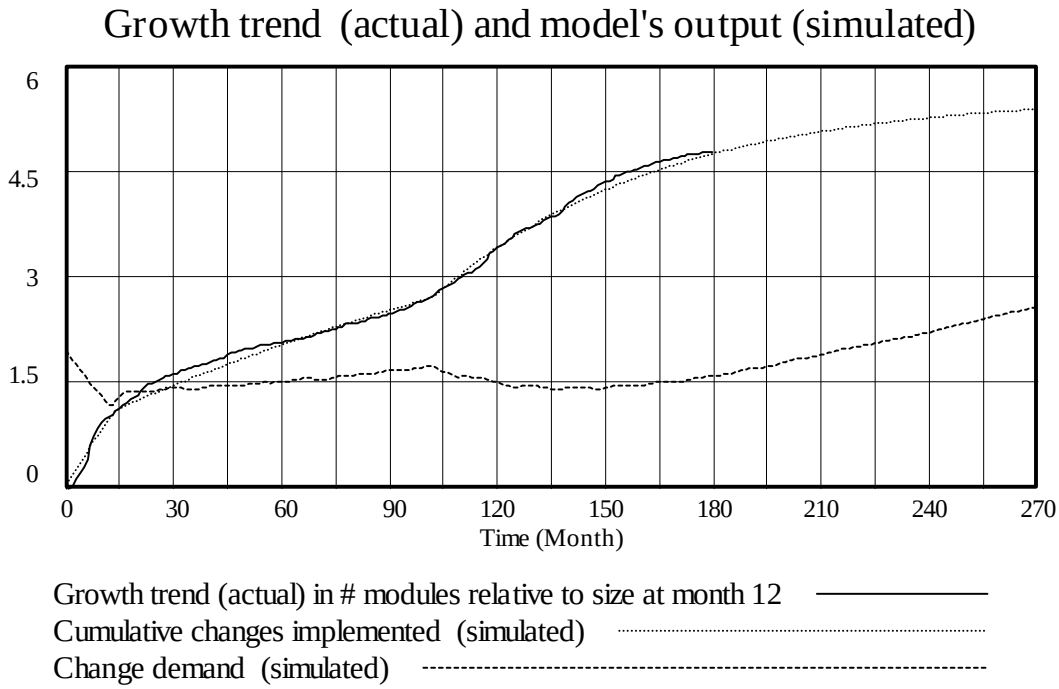

Figure 3 - Simulated process behaviour with no resources assigned to anti-regressive work and with all three variables relative to the actual size of the system at month 12

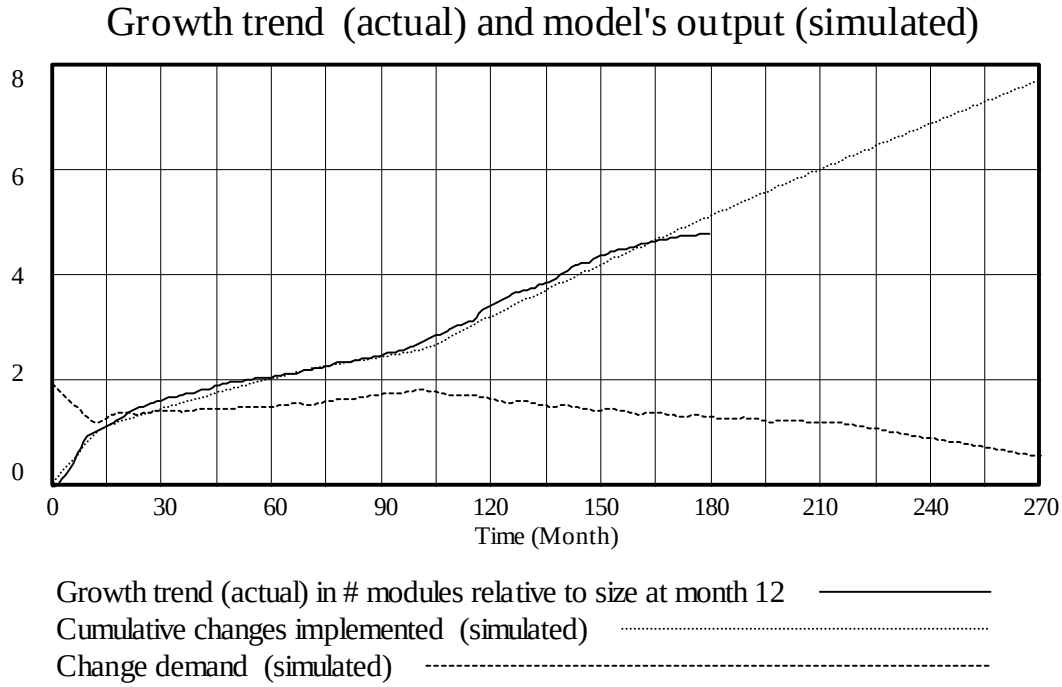

Figure 4 - Simulated process behaviour with 30% of the resources applied to anti-regressive work and with all three variables relative to the actual size of system at month 12

APPENDIX

Definitions of the Variables in the Model

Variables Related to the Technical Software Process

Change request – Represents changes to requirements currently being implemented or already part of software's functionality. It also includes incoming new requirements.

Exogenous rate – Refers to the flow of change requests whose origin cannot be associated with adaptation of already implemented requirements. It has been modelled as a random process Poisson[1] which is frequently used, in general, to represent request arrivals to a servicing queue.

Change demand – Changes waiting to be implemented but not yet released.

Rate of change completion – Refers to the rate of completion of changes including *all* necessary activity, such as implementation, quality verification and validation (QV&V), integration and system validation, etc.

Changes implemented – Changes implemented but not yet released. This is consistent with the observation that completed changes are released to the final users in batches. This practice may be associated with reduction of system integration and validation costs, optimisation of resources involved in the latter, need for simplification of user documentation updates and training of support personnel, need for minimisation of costs of reinstallation and/or due to contractual, administrative or policy-related constraints (e.g. when periodic releases at fixed intervals are the case).

Cumulative changes released – Changes already released as part of a version or release of the software. This represents the functionality already delivered to the users.

Release policy – Determines the points in time at which the completely changes are effectively released to the users.

[1] D. Williams suggested us the use of the Poisson distribution.

Variables Related to the Usage Domain

Stakeholder response to change, multiplier of work requests and *Stakeholder response to change, delay*

With regards to these part of the model a *ripple* effect in released functionality is postulated, that is, after changes are released to the users it is expected that need for further adaptations and customisations will arise. This mechanism is modelled using a 3rd order delay (Coyle, 1996), which in Vensim (Vensim, 1995) is invoked by the command:

Output = delay3(input,delay)

and in the model is implemented as:

Endogenous rate = delay3(STAKEHOLDER RESPONSE TO CHANGE multiplier of work requests

**Rate of release of changes,STAKEHOLDER RESPONSE TO CHANGE delay)*

Variables Describing the Resources Applied to Evolution

Anti-Regressive effort – As defined by Lehman [leh74], effort devoted to activities which do not increase directly the power, function or performance of the software, but which will enable further evolution. Examples of these are software and documentation re-structuring and rewriting for complexity reduction, elimination of dead code, etc.

Progressive effort – Development effort addressed to increase power or function, or to improve software performance.

Average ab initio *individual productivity* - Average number of progressive changes implemented by a person in one month when working in an *ab initio* project.

Team size – Total number of people assigned to the evolution team at any given moment in time. It is assumed that all the members are of average experience and productivity. A further refinement of the model might include the personnel training and learning dynamics.

Staffing policy – Number of person-months assigned to the evolution team over software lifetime. Team size is directly derived from Staffing Policy. The effects of different staffing policies can be examined.

Cumulative changes implemented – Total number of changes implemented over system lifetime.

Cumulative anti-regressive changes – Total number of anti-regressive changes implemented over system lifetime.

System type – A calibration constant. It reflects the impact of the neglected anti-regressive work in progressive productivity. The latter is defined as the difference between the cumulative changes implemented and the cumulative anti-regressive changes.

Progressive work productivity – Number of changes implemented per person-month. In the model it is given by:

Progressive work productivity = *AVERAGE ab initio INDIVIDUAL PRODUCTIVITY*effect of team size*effect of neglected anti-regressive work.*

The formulation in Vensim (Vensim, 1995) is the following:

Progressive work productivity = AVERAGE ab initio INDIVIDUAL PRODUCTIVITY((TEAM SIZE^0.2) - ((1/1800)*TEAM SIZE^2))**

(1-SYSTEM TYPE(Cumulative changes implemented-Cumulative anti-regressive changes))*

The effect of the team size is modelled by two terms. The first represents the effect of a mutually co-operating group of people and hence gain in productivity as the team increases its size. The second reflects losses due to interpersonal communication overhead, needed to coordinate the work of large groups of people. The second term is calibrated such that in a group of 30 people working in a closely co-operative fashion 50 percent of work would be lost in meetings and communications, as suggested in (Adbel-Hamid and Madnick, 1991). This is exemplified in Fig. A.1, which shows the productivity function and its constituents, with maximum team productivy at a team size of approximately 20 people. It also shows that given such formulation, the total team output per unit of time reaches a maximum at 40 people. Adding people to a team of that size will reduce the total team output. Such a generic equation offers the basis for a productivity function, once calibration of its parameters to a particular process has been performed.

Productivity threshold - Desired progressive productivity.

Anti-regressive work productivity - Number of anti-regressive changes implemented per person-month. In

the model it is given by:

Anti-regressive work productivity = AVERAGE ab initio *INDIVIDUAL PRODUCTIVITY*((TEAM SIZE^0.2) - ((1/1800)*TEAM SIZE^2))*

Note that the same formula reflects changes in progressive work productivity due to team size.

Anti-regressive work policy - A parameter between 0 and 1 to reflect the percentage of the Team Size to be dedicated to this type of work if the policy is activated.

Figure belonging to Appendix of the paper

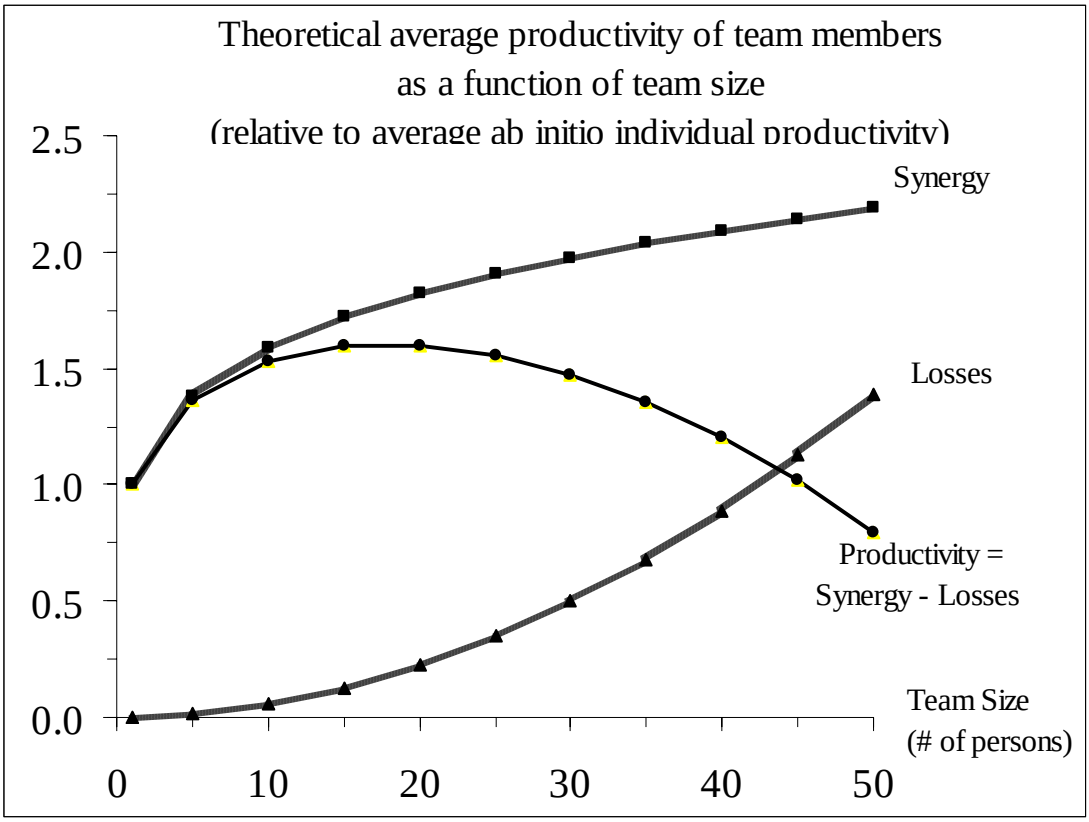

Figure A.1 - Theoretical average productivity of team members as a function of team size