System Dynamics Modelling for the Management of Long Term Software Evolution Processes

Goel Kahen Meir M Lehman Juan F Ramil
Department of Computing
Imperial College of Science, Technology and Medicine
180 Queen's Gate, London SW7 2BZ
tel +44 207 594 8216; fax +44 207 594 8215
{gk, mml, ramil}@doc.ic.ac.uk

Abstract

An approach and basic concepts for the study of the *system dynamics* of long-term software evolution processes is presented. The approach provides a generic context and framework that supports at least three crucial process areas requiring management decision, resource allocation, release planning, and process performance monitoring. The report exemplifies the approach with an executable model. The latter reflects the global software process at a high level of abstraction and includes phenomenological observations derived from the laws of software evolution and the behaviours thereby implied. It incorporates concepts such as *progressive* (e.g., functional enhancement) and *anti-regressive* (e.g., complexity control) activities and enables the study of policies of human resource allocation to classes of activities. The example shows how the model permits assessment of the impact of alternative policies on various evolutionary attributes. It is part of and exemplifies the methods for software process modelling being developed and applied in the FEAST projects.

Keywords: Management, metrics, process modelling, effort allocation policies, process improvement, release planning, release management, anti regressive, progressive, software process, system dynamics, software evolution, white box models

1 Introduction

This report presents and discusses an approach for the planning and management of *E*-type (Lehman & Belady 1985) long-term *software evolution* processes and exemplifies it with a model that is used in the evaluation of effort allocation policies. The approach addresses two phenomena, the *continuing change and enhancement of functional content* and the *growing complexity* of the software product, that are believed to be important in the evolution context. The present approach and model, together with earlier models (Lehman & Wernick 1998; Wernick & Lehman 1999; Chatters *et al* 2000; Kahen *et al* 2000) exemplify the FEAST white box investigation into the dynamics of evolution processes. With regards to the present model the next step is to align it with actual industrial processes. If successfully accomplished, the result would be models calibrated to specific processes to be used in their planning, management and improvement. Once this is achieved, a *generic* model may emerge. The latter would be a useful tool for subsequent tuning to specific systems and evolution domains, and thereafter for direct use in the context of industrial processes for process improvement and software evolution planning and management.

2 Evolution Planning and Management

The approach and the model presented here, a refinement of our previous system dynamics (Forrester 1961) modelling work (Lehman & Wernick 1998; Wernick & Lehman 1999; Chatters *et al* 2000; Kahen *et al* 2000), is offered, *inter alia,* as a contribution to the more general issue of software evolution planning and management. This is an area that poses a number of unanswered questions. These include, for example:

- how, in general, should a software organisation plan for the future by exploiting past evolutionary experiences and historical data on evolving software?
- how may the business knowledge and understanding that developers and managers acquire be used more effectively in helping them perform and manage software evolution processes?
- how can software managers cope with the difficult strategic changes that must be overcome in order to sustain software system functional growth in changing business environments?

© G Kahen, MM Lehman and J.F. Rami. All rights reserved.

- how are limited resources (mainly, human effort) best allocated to different types or classes of activity?

Since in industrial settings the software evolution process is embedded in various socio-cultural, organisational, cognitive and economic environments, a full answer to the above questions is not straightforward. The difficulty is compounded by many factors. One would expect that with, perhaps, the exception of the less *mature* processes and of software development in-the-small, software evolution, like other complex processes, be characterised by:

- process attributes that exhibiting *non-linear* behaviour. Such behaviour tends to be difficult to predict and control.
- *tightly coupling* between evolutionary attributes. Everything appears to influence everything else.
- *presence of strong system dynamics*. Dynamic effects take place on different time scales and at many levels of aggregation.
- *counter-intuitive behaviour*. Cause and effect are distant in time and space, and effects may persist long after their causes have ceased to operate. This makes effective management very difficult (Forrester 1971)

To the above challenges one may add at least one more. The processes are performed and managed by humans whose individual decision-making behaviour is essentially unpredictable. One would therefore expect it to be unlikely for the process to display regularities or invariants. That is, the process will, at every instant in time, be the result of and reflect local decision making in the context of locally perceived circumstances. Nevertheless, a series of empirical studies of evolution processes over the years, most recently as part of FEAST/1 and /2 (FEAST 2000), have assembled evidence that suggests that evolution processes ***do*** tend to exhibit regularities, invariants and, in addition, commonalties. They are briefly referred to in the next section. What is important here is that they offer a basis to build models that can facilitate long-term evolution planning and management, as exemplified by the model presented later in this report.

3 Software Process and Evolution Laws

A series of investigations into the phenomenology of software[1] evolution have been carried out over a period of some 30 years or so (Lehman 1969; 1974; Chong-Hok-Yuen 1981; Kitchenham 1982; Lehman & Belady 1985; FEAST 2000). The studies have resulted in the recognition of a set of regularities that tend to emerge, with varying emphasis and to a varying degree in evolutionary attributes of *E*-type (Lehman & Belady 1985) software and their processes. The type *E* refers to software being actively used in solving a problem within a real world domain[2]. Most systems upon which businesses and organisations rely for their operations are of this type.

Behavioural patterns and invariant behaviour have been identified over the years by observations derived from metric data of the several systems studied and have been encapsulated in eight *laws of software evolution* (Lehman 1974; 1978; Lehman & Belady 1985; Lehman 1989, FEAST 2000) and a *principle of software uncertainty* (Lehman 1989). More recent evidence supporting them has been summarised in a series of papers available from links at the FEAST web page (FEAST 2000). Table 1 (next page) shows the laws as re-formulated in August 2000. The laws are numbered and presented in the order of their identification. Column one indicates the year in which the phenomenon was recognised. The eighth law provides a formal statement of the feedback system nature of *E*-type evolution processes. The other seven encapsulate various aspects of inter-acting feedback influences. That is, it seems plausible to hypothesise that the eighth

[1] The reference to *software* applies here to the product of the software process as a whole, that is, it includes programs (source code), development and usage documentation and other documents such as specifications, design, test plans and cases, etc.

[2] All references to software and software process in the remainder of this report are to this type.

© G Kahen, MM Lehman and J.F. Rami. All rights reserved.

law is the source of mechanisms (or forces) that underpin the other seven.

The laws offer an initial identification of behavioural invariants and a statement of their expected qualitative behaviour. The most recent investigation of evolution phenomenology, the FEAST/1 (1996-1998) and FEAST/2 (1999-2001) projects was motivated by the study of the FEAST hypothesis, which includes the eight law as one of its assertions. The hypothesis, in one of its formulation is as follows: *"E-type evolution processes are multi level, multi loop, multi agent feedback systems and must, in general, ,be treated as such to achieve major process improvement for other than the most primitive processes."* (Lehman 1994).

The work and model presented in this report form part of the FEAST/2 project and of the investigation of the hypothesis. The model encapsulates phenomena such as continuing demand for change (first law) and continuing growth (six law) implied by the laws. As will be seen later in the report, the model portrays the software evolution process as a feedback system with a basic loop in which the process output, the operational software, is fed back to the process input.

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	Global *E*-type system evolution processes are self-regulating
IV 1978	Conservation of Organisational Stability	Average global activity rate in an E-type process tends to remain constant over periods or segments of system evolution over its lifetime
V 1978	Conservation of Familiarity	In general, the incremental growth and long term growth rate of *E*-type systems tend 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

Table 1: Current Statement of the Laws - Sept. 2000

4 Objectives of White Box Software Modelling in FEAST

System dynamics (Forrester 1961) and soft system methodology (Checkland 1981; Checkland & Scholes 1990) represent two of the techniques that have been proposed to model real world complex processes. Application of the system dynamics modelling approach, in particular, enables one to explain the behaviour of a complex system in terms of its feedback loop structure, often called the *systemic* structure. That is, system dynamics is a powerful tool that, when properly applied, can help software managers to deal with the *systemic* properties of their decision environments. Note that, from the start, the field of system dynamics was conceived and applied to examine the *behaviour* of complex social and organisational systems through computer modelling and simulation tools (Meadows *et al*, 1972) such as the Vensim tool (Vensim 1995) used to develop and execute the model presented in this report.

The focus of the system dynamics field is on the modelling of the feedback structure of the system involved, and in doing so, to promote a better *understanding* of the system and improve decision making. The significance of this observation is better recognised when one takes into account, for example, the empirical evidence from experiments with subjects in

© G Kahen, MM Lehman and J.F. Rami. All rights reserved.

simulated environments. This seems to indicate that failure to foresee, take into account and effectively oversee feedback effects is one of the sources of management error or ineffectiveness. It is also evidenced, for example, by studies which indicate that management dysfunctional behaviour in complex systems may be explained by systematic errors made by the decision makers in failing to account for feedback, time-delays and non-linearities (Senge 1990; Kleinmuntz 1993; Sterman 1994; Langley 1998). Studies conducted with students in the software engineering field point towards overall similar conclusions (e.g., Drappa & Ludewig 2000).

This report provides one answer to the question of how to approach planning and management of a software evolution process by presenting a model that differentiates between various classes of resources (i.e., effort) within the systemic structure of the software process. This is explained later in this report. By exploiting system dynamics modelling technology the model described seeks to reveal and explain the dynamics of long-term software evolution within a model of the process that articulates the evolutionary behaviour of a software process as a function of different resource allocation policies.

The basic model developed in this report seems to be, in principle, applicable to Waterfall-based (Royce 1970) traditional single-team *E*-type evolution processes[3]. It has been built with the following intentions in mind:

- to capture relevant aspects common to all such processes in the model
- to replicate these, at a high level of aggregation, in the context of the behaviour of software processes
- to avoid the inclusion of elements believed not to be common to such software processes.

[3] Model adaptation to other evolution processes such as *build & fix* (Schach 1990), multiple teams and parallel work, distributed evolution teams, components-intensive, etc. may be achieved at a later stage. Such adaptation may be accomplished as part of the confrontation of the model with industrial processes.

Development of the model has generally followed a top-down successive refinement process (Zurcher & Randell 1968; Wirth 1971) a key feature of the FEAST white box modelling approach (Kahen *et al* 2000), as exemplified in the earlier system dynamic models (Lehman & Wernick 1998; Wernick & Lehman 1999; Chatters *et al* 2000). In so doing it has been necessary to exclude some of the real world elements, particularly those believed not to be common to all processes considered. Other influences believed to yield second-order effects have been also excluded. The fine-grain detail of a given process can be reflected in the model at a later stage, for example, when a model is tailored to a particular software process. That is, the model offers a template that can be customised to individual processes. In this context it should be noted that the immediate focus of interest of many of the existing system dynamics studies of the software process (e.g., Abdel-Hamid 1991) appear to have been based on single step development of fine grained models (e.g., a bottom-up approach). Such an approach results in large models that, though possibly of value within the context of the individual process they reflect, tend to be cumbersome. They also cannot be easily ported from process to process, appear difficult to calibrate, interpret and exploit, and have not found widespread acceptance as decision-making tools.

A system dynamics model can reflect a system at many levels. It is important to identify an appropriate level of abstraction or aggregation that results in a model that is appropriate for the purpose for which it is being constructed. Within such a model, one may proceed to identify the main feedback loops relevant to the issue or problem of interest. But, by and large and as recognised by other workers in the field (Ruiz & Ramos 2000), modellers have chosen to represent the systems at a too low level of abstraction.

It should also be noted that 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* 1993

© G Kahen, MM Lehman and J.F. Rami. All rights reserved.

and the FEAST white box modelling work (Lehman & Wernick 1998; Wernick & Lehman 1999; Chatters *et al* 2000), has the *global* process (Lehman 1994) and long-term evolution as their focus.

5 The Approach

The determination of appropriate evolution policies shares with the general optimisation problem the following five classes of attributes as illustrated in Figure 1:

- *objectives,* also termed goals, are the variables or expressions to be optimised
- *constraints* are conditions, such as the ranges within which some of the model variables are forced to remain
- *structure* includes behavioural invariants, causal mechanisms, feedback loops, built-in process relationships that are perceived as properties of the process
- a *policy* represents a "...guiding rule which would be applied continually as time passed and circumstances changed..." (Coyle 1996, p. 222) and that need to be formulated to achieve the objectives under the constraints and under the behavioural invariants. Coyle introduces a distinction between *policy* and *pressure point*. The latter imply "...a choice which is made only once, after which the system will run under the influence of that choice..." (Coyle 1996, p. 222). Here both concepts are considered under *policy*.
- *Behaviour* emerges from the interaction of the above. Policies are satisfactory to the extent that the model's behaviour is consistent with organisational objectives

The figure illustrates this view of the problem with examples of the individual constituents. The solid-line boxes represent the major constituents of the system dynamics model described in this report. Possible objectives of the modelling exercise are indicated within the dashed-line box.

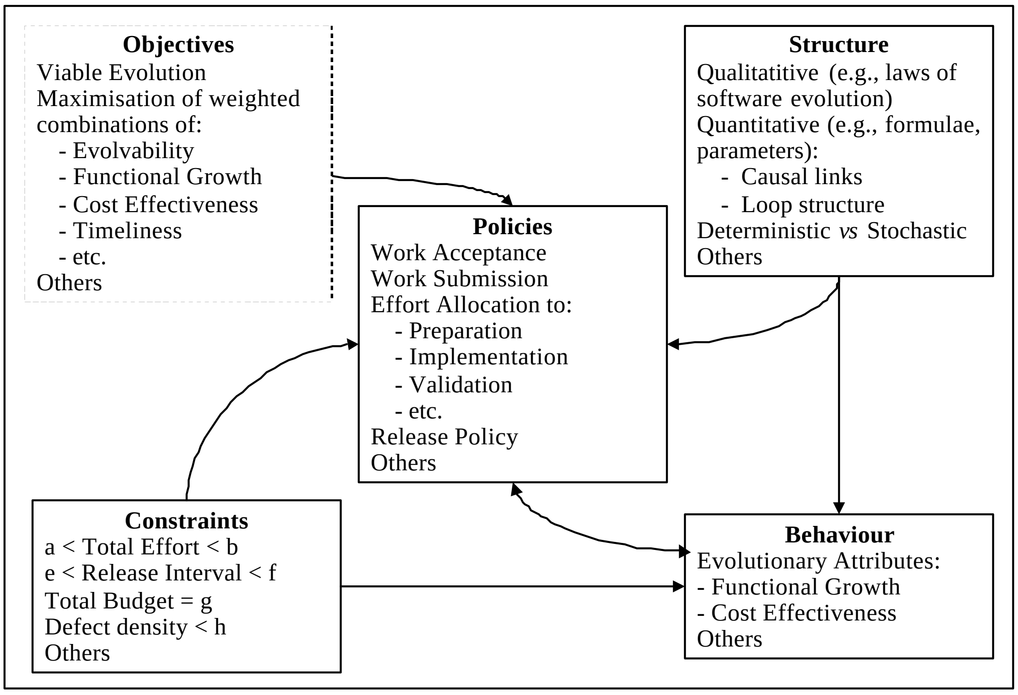

Figure 1: Model constituents (solid-line boxes) and possible objectives to be achieved (dashed-line)

The goal of a modelling exercise involving the elements in the figure would be to identify policies that lead to viable evolution as defined by organisational objectives and, within this, to the comparison of the effects of different policies. Other objective functions are not examined, since it appears that they would, to a large degree, be dependent on the evolution domain.

It is, of course, not only objectives that vary from organisation to organisation and from process to process. How objectives are

© G Kahen, MM Lehman and J.F. Rami. All rights reserved.

promulgated, applied and enforced will also vary. Thus, even placeholders appear inappropriate in the present model. The necessary additions have been left to subsequent refinement and adaptation of the model to a specific domain and process.

Note that in the figure *viable evolution* serves to define a reasonable set of evolutionary attributes within reasonable boundaries as perceived by the various evolution stakeholders. This includes, *inter alia*, continuing process cost-effectiveness and successive release of software version with increased functional power. These must, at the very least, satisfy the evolving needs and desires of mandatory, essential or otherwise selected stakeholders.

In figure 1, the arrows in the figure must be interpreted as follows: the constituents in the category being pointed to are generally a function of the constituents in the category from which the arrow originates. Arrows at both ends indicates a two-way dependency. Thus, for example, *Behaviour* must depend on *Policies*, but equally *Policies* may have to be *adapted* as a result of behaviour.

Examples of policies in the model are those that regulate the progress of a process through activities such as work acceptance, implementation, validation and release activities. The interrelations amongst model constituents are governed by causal relationships and by the effect of feedback looks. Only influences that may change significantly over system lifetime can be considered in the model. Organisational, social, economic, and other influences that do not often vary significantly over the period studied are considered constant and not explicitly represented in the model. If and when they change significantly, this could to be treated as *structural changes*, and treated as changes in model equations and parameters.

The system dynamics model in this report presents the basis for the integration of the above mentioned elements in a rational and coherent way which is aimed at supporting planning and management of long-term evolution processes.

6 Change and Complexity in Evolving Software and their Processes

The basic evolutionary feedback loop operates as follows. Demand for change and evolution are in a mutual relationship. Evolution arises from the need to maintain user satisfaction within a changing (application, usage) domain. This is reflected by such factors as changes in the application domain, changes in user preferences, learning and experience, human ambition and the influence of agents and factors exogenous to the application and system. The act of releasing the operational software in the application domain is likely to accelerate the rate of change of the latter (Lehman & Belady 1985).

One aspect of this phenomenon is reflected by the change of the *application concept*. The latter represents the desires of users or potential users and other stakeholders with regards to the functional content at a given moment in time. Evolution is viewed as the process that changes the software by changing existing functionality and adding new capability, so that the former and the system as a whole are adapted and enhanced to address changes in the application concept. The ultimate goal is to maintain or increase stakeholder satisfaction within a changing application domain, and changing viewpoints and desires of the stakeholders. Some of the changes will be triggered by previous changes. For the reasons discussed in the previous section, we do not comment here on such roles and interests as those of the business and other groupings within which the process is embedded.

Within the situation depicted in the previous paragraph, identification and timely satisfaction of the demand is a major challenge in the software evolution process. Overcoming this challenge has at least three aspects. Firstly, due to the consequences of adding change upon change upon change, the software becomes more and more complex unless work is done to compensate for it (Law II). Complexity also increases because of the need to add functionality. The latter is reflected in the increasing size of software

© G Kahen, MM Lehman and J.F. Rami. All rights reserved.

(Law VI). Another source of complexity growth is the orthogonality of new function to system architecture. Secondly as, for example, user integration with and dependence on the system becomes ever greater, time constraints on change implementation tend to become increasingly tighter. Thirdly, each change may, to certain extent, trigger more demand for change. This leads to a positive loop that may severely impact the implementation time and completion delay.

A consequence of the above is the tendency to perform parallel work This is likely to increase process complexity *per se* and lead to hidden decisions and other errors. All these phenomena represent factors that lead to product and process degradation, to excessive expenditure or to all these. It may also result in a need for additional effort to counteract or control such effects, as explained below. When such work is neglected, more time and effort per change unit implemented is required than would otherwise be needed. In principle, any single change in one stage or in an element of the software (or its process) may cross-impact on other stages and elements, creating multiple ripple or cascade effects across the system. In the context of evolution, with continuing change and increasing functionality, these is likely to lead to increased rework and delay, and hence to declining productivity. This would explain, at least in part, the decrease in evolution rate over time observed in several systems and captured by the *inverse square* model of system growth proposed by Turski (1996) and further explored in FEAST (2000).

Managers can take cognisance of the above influences. By taking them into account, they may then direct effort to specific activities that otherwise would have been ignored or neglected. They may be able to *control* key evolutionary attributes. This would be part of a discipline of long-term evolution planning and management. The latter appears to be a necessity in a world in which organisations increasingly rely on growing (and aging) *inventories* of evolving software for their operation.

7 Progressive and Anti Regressive Work

In general, maintenance and evolution activities have been classified as functional fixing, adaptation, and enhancement of existing functionality. Several classifications of maintenance and evolution work have been proposed over the years (Chapin *et al* 2000). The idea is that improved understanding and management of the process can be achieved by classifying and measuring each of the activity classes. We refer here to one that focuses on the growing complexity phenomenon issue**,** as suggested by one of the present authors (Lehman 1974)**.**

As discussed above one effect of change upon change upon change is increasing complexity of the software and other aging effects (Lehman and Belady 1985, Parnas 1994). These must, if not adequately compensated for, lead to a decrease in evolution productivity. Following Baumol's classification (Baumol 1967) of work effort into *progressive* and *anti-progressive* types, Lehman suggested a further category, *anti-regressive* (Lehman 1974, 1985). Activities that enhance (by addition or modification) system functionality were termed *progressive.* The new term anti-regressive was used to refer to work effort intended to compensate for the *aging* effects (Parnas 1994). Such work consumes effort without any *immediate* visible stakeholder return, for example, in system value as reflected by system functional power or performance. Instead, it facilitates further evolution, enabling it to be achieved more easily and requiring less effort. When a new release of a software system is developed, it improves previous releases and becomes structurally more complex unless the anti-regressive effort is sufficiently large to compensate for such complexity growth. There is, therefore, a need to allocate more anti-regressive effort. Growing complexity is reflected by increased size, more functionality, a larger number of integrated components, more control mechanisms, a higher level of reciprocal interdependency and greater inter-element connectivity. In this context, the achievement of a minimum level of anti-

© G Kahen, MM Lehman and J.F. Rami. All rights reserved.

regressive activity is seen as essential to ensure further evolution.

The remainder of the report introduces a model that exemplifies the approach and incorporates the above concepts.

8 Influence Diagram

In order to understand the dynamics of the situation and possibly overcome the challenge, one needs to study the overall feedback loop structure which, it is believed, determines to a significant extent the overall process performance. To do this, major process constituents need to be reflected in a model in which the individual forces are quantified and the behaviour implied by their combined effect can be derived and studied.

As for other systems, the structure of a software evolution process can be sketched using *influence diagrams* (Coyle, 1996). Causal loops can be either balancing (i.e., negative feedback) or reinforcing (i.e., positive feedback). A balancing loop in isolation exhibits goal-seeking behaviour. This means that after a disturbance, the system seeks to return to an equilibrium situation. A reinforcing loop in isolation, on the other hand, drives exponential growth or decay. An initial disturbance leads to further change, and may build up and result in instability.

The model of figures 2 to 4 has been developed incrementally from the laws of software evolution, analysis of previous research (FEAST 2000), preliminary field work with FEAST/2 collaborators and a study of how others have approached the development of system dynamics models of the software process. Detailed analysis of all these led to the high-level causal loop diagram shown in Figure 2. In the diagram the solid lines represent either positive '+' or negative '-' influences between two of the attributes. Dashed lines represent influences that, though real, are considered more complex than simply positive or negative. Figure 2 models a closed loop process that is consistent with the feedback system view studied in FEAST (2000). Amongst the attributes indicated in the figure there is a set of policies. One needs to assess their individual role and to address how to harmonise them to achieve process understanding. This, however, requires a more detailed executable model. That is, due to difficulties of behavioural analysis and, concurrently, reaching understanding of the dynamic behaviour of feedback loops, we need to move from causal loop diagrams to more formal models which can be simulated.

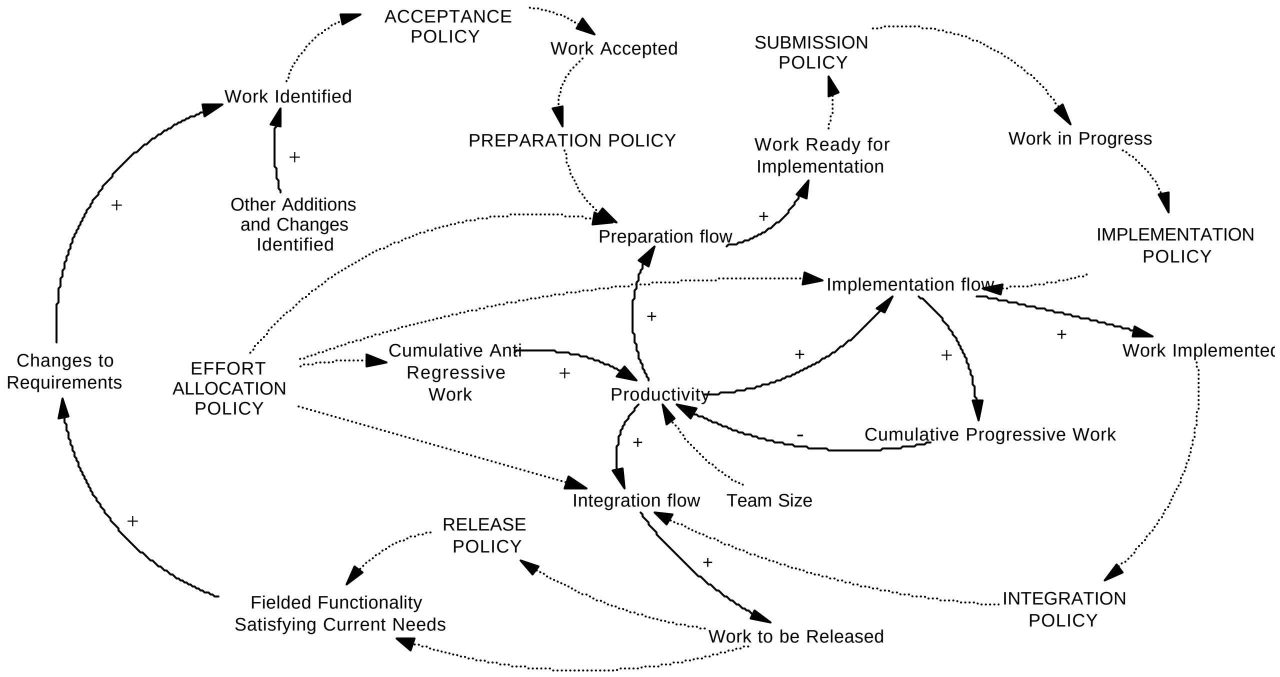

Figure 2: Influence diagram of the software evolution process

© G Kahen, MM Lehman and J.F. Rami. All rights reserved.

9 Full Model

As already indicated system dynamics techniques provide one convenient approach. In general, such models consist of two components: the stock and flow network, and the information network. The causal loop diagram of Figure 2 has been operationalised and has now been developed into a fully specified and executable model. A *portion* of the model diagram is shown in Figure 3. For clarity and simplicity of presentation some details of the structure have been omitted and will be shown later in the report. Starting at a high-level of abstraction, the figure presents a simplifed view of the evolution process. The latter is visualised as a process that addresses a continuing flow of work. The simplified view as generated by the Vensim tool is displayed in Figure 3 (Vensim 1995). Levels or stocks are indicated by variables within the rectangles. The double-line arrows represent flows or rates. The single lines indicate that one variable has an influence in determining the other.

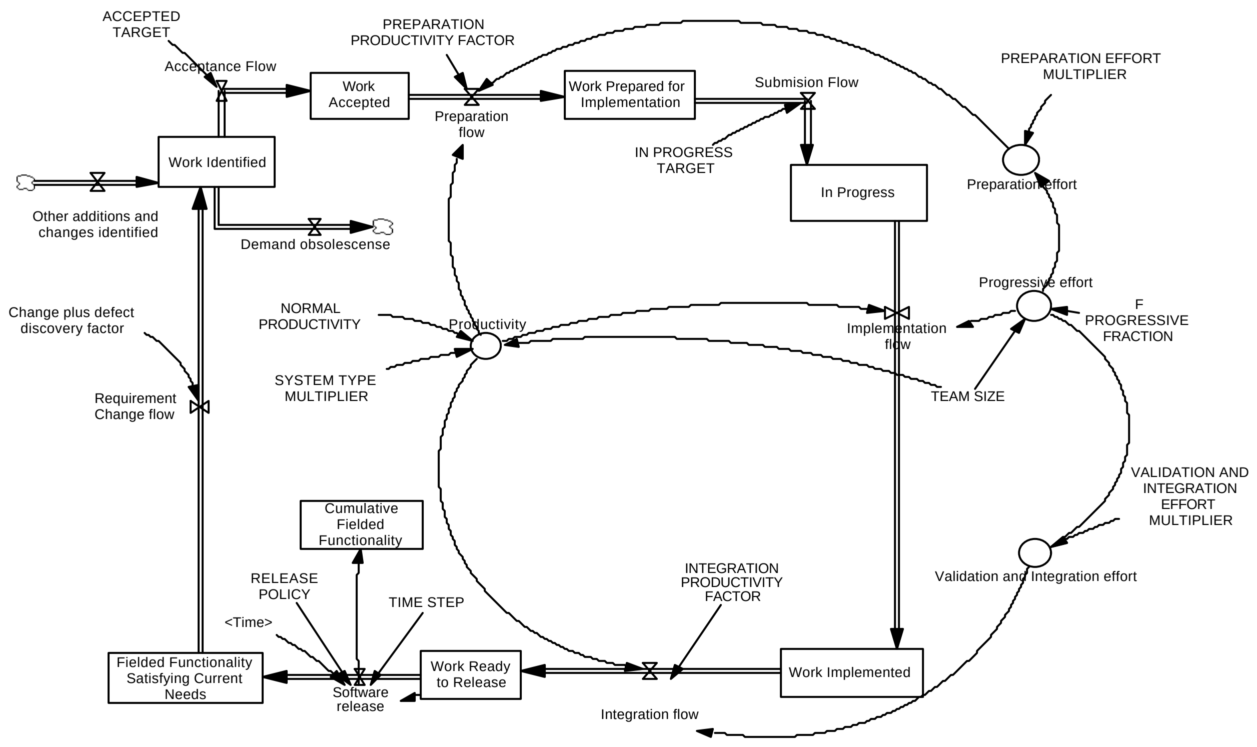

Figure 3: A simplified model diagram of the software process dynamics

In general, *Work Identified* (or work demand) encompasses both, new functionality, and fixing and modification of existing functionality. For the sake of simplicity further distinction between the last two categories is not made here, though this may be necessary as the model is further refined. The model considers two classes of sources for work: *Requirement Change flow* and *Other Additions and Changes identified*. The first class stems from functionality already in the field that requires fixing, adaptation or amendment. The latter class represents new functionality. A demand obsolescence phenomenon, represented by work requests that were not accepted and are no longer needed, is also included. Five different level variables represent technical process: *Work Accepted, Work Prepared for Implementation, Work in Progress, Work Implemented* and *Work Ready to Release*. An important policy parameter is represented in figure 3 by the fraction of the personnel dedicated to pursue progressive activities, as represented in by the *F PROGRESSIVE FRACTION* on the right-hand side of the figure. This is reflected in the model by the expression:

Progressive Effort = F PROGRESSIVE FRACTION * TEAM SIZE

© G Kahen, MM Lehman and J.F. Rami. All rights reserved.

Once the progressive effort has been selected by means of the parameter *F*, the model assigns a fixed proportion of such effort to *Preparation Effort* and to *Validation and Integration Effort*. That is, it is assumed that per each unit of progressive work there is a fixed amount of preparation and validation work to be applied. Such policy is implemented by the expressions:

Preparation effort = (Progressive effort *PREPARATION EFFORT MULTIPLIER)

Validation and Integration effort = (Progressive effort *VALIDATION AND INTEGRATION EFFORT MULTIPLIER)

10 Details View and an Example of Policy Evaluation

The more comprehensive schematic of the system dynamics model is displayed in Fig. 4. This view includes all variables and parameters within the current executable Vensim model. Equations that form part of the executable model are included in the Appendix that presents the full model in Vensim language (Vensim 1995).

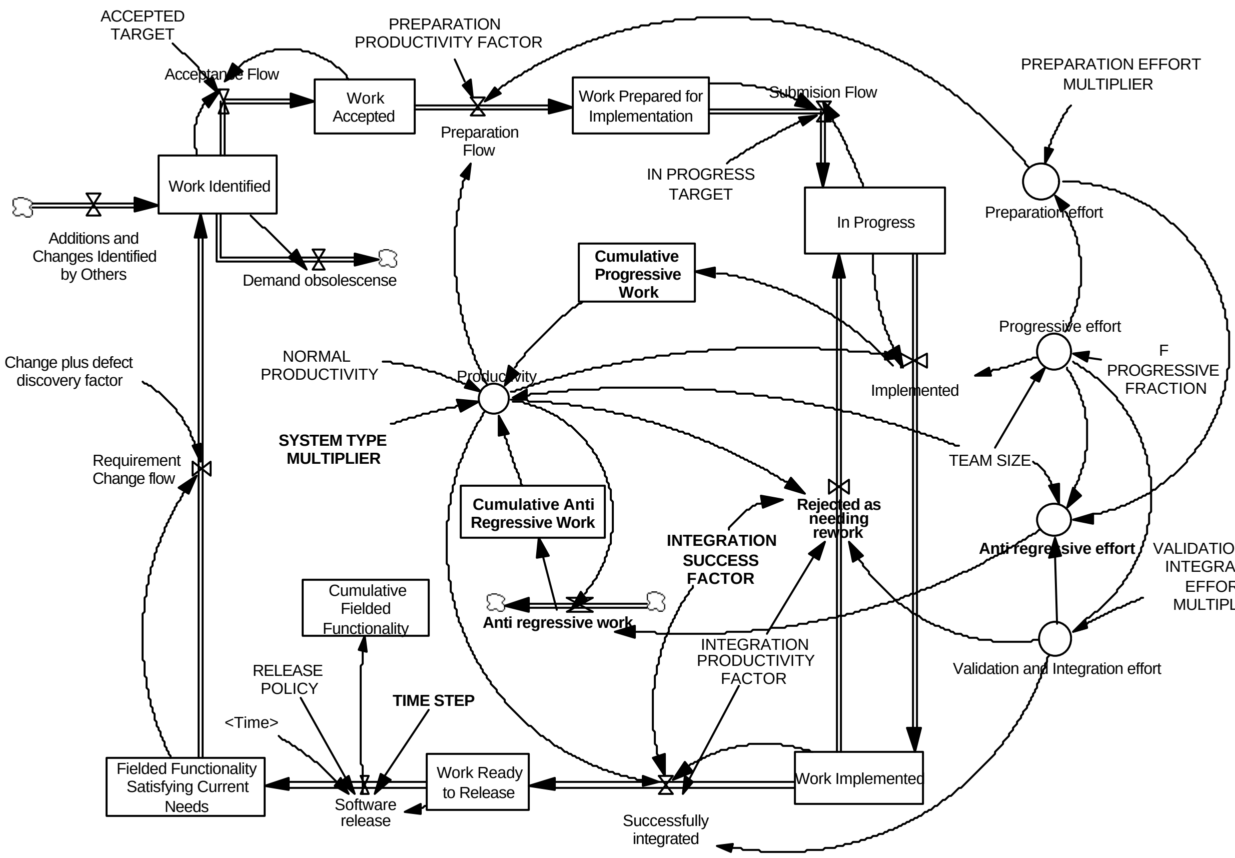
ACCEPTED TARGET
Acceptance Flow
Work Accepted
PREPARATION PRODUCTIVITY FACTOR
Preparation Flow
Work Prepared for Implementation
Submision Flow
PREPARATION EFFORT MULTIPLIER
Work Identified
Additions and Changes Identified by Others
Demand obsolescense
IN PROGRESS TARGET
In Progress
Preparation effort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

Figure 4: Detailed view of the system dynamics model

A detailed explanation is left for a more comprehensive report.

As an extension of figure 3, figure 4 incorporates additional elements, shown in bold, such as the variables involved in the application of effort to anti regressive activity. It also includes a rework feedback loop between *Work Implemented* and *Work in Progress*. Such rework is considered here as implemented work that has failed validation testing, for example. The amount of rework in the process is controlled by an *INTEGRATION SUCCESS FACTOR*, between 0 and 1. Note that for the sake of clarity and simplicity, other such feedback 'rework' loops are left to be considered and possibly included when the model is applied to particular industrial processes, after having found that such loops are indeed significant. The remainder of the discussion focuses on the issue of assignment of effort to the different activity types

© G Kahen, MM Lehman and J.F. Rami. All rights reserved.

considered and its effects on long-term evolutionary attributes.

The model in figure 4 embeds the assumption that once the progressive, the preparation and the validation portions of the available effort have been assigned (as explained in the previous section), the remaining effort (if any left) is allocated to anti regressive effort. This is represented in the model by the expression:

```
IF (TEAM SIZE-Preparation effort-Progressive effort -
    Validation and Integration effort)>0
THEN
    Anti regressive effort =(TEAM SIZE-Preparation effort
    -Progressive effort-Validation and Integration effort
ELSE
    Anti regressive effort=0)
```

Thus, when, for example *F* is 0.5 or 50 percent, and the *Progressive Effort Multiplier* and the *Validation and Integration Effort Multiplier* are also 0.5, no effort would be left for anti-regressive activity. Thus, in this case 0.5 would be the maximum value for the policy variable *F*.

One interesting investigation is to explore the long-term consequences of different values of *F*. Figure 5 show the results of model execution in which three different values of the parameter *F* are applied, keeping *TEAM SIZE* and the *Preparation and Validation and Integration Effort Multipliers* fixed (as 0.5). In the figure, F05 indicates the parameter *F* equal to 0.5, F03 indicates 0.3, and F02 indicates 0.2. The other parameter values remained fixed during the particular set of runs illustrated here.

.

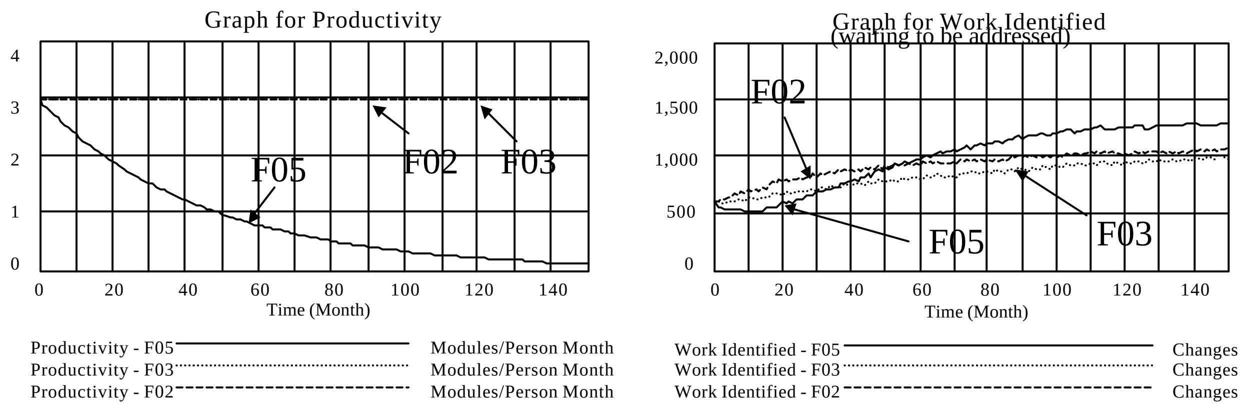

Figure 5a,b: Examples of Vensim model output

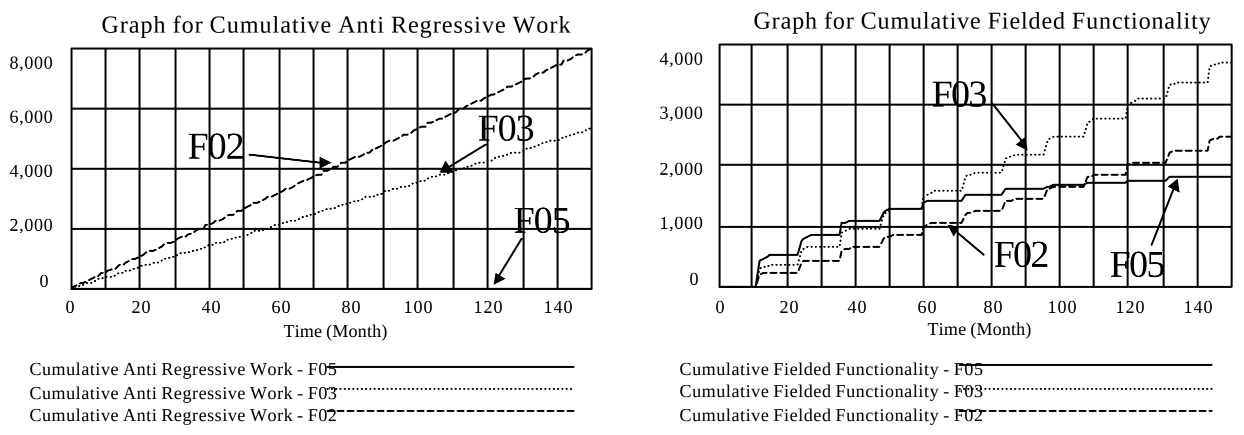

Figure 5c,d: Example of Vensim model output

The figure shows how different allocation policies yield different effects in attributes such as *Productivity, Work Identified, Cumulative Anti Regressive Work* and *Cumulative Fielded Functionality.* Note that the 0.3 and 0.2 values provide constant productivity over the entire 150 one-month periods included in the model run illustrated.

© G Kahen, MM Lehman and J.F. Rami. All rights reserved.

This is due to the high level of anti-regressive activity that this fraction permits. The overall results of the runs illustrated by Figure 5a,b,c and d suggest that the intermediate fraction (0.3) yields the highest level of cost effectiveness over the long term (i.e., highest cumulative fielded functionality) and also provides further potential for growth. A value of F less than 0.3 or so, as shown in the F02 case, though leading to a worst performance is still in some respects better over the long term than F05. This exemplifies that anti regressive activity, at least in the regime dictated by the policies implemented in the present model, leads to higher effective productivity despite the fact that it takes resources away. 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.

11 Discussion

The model presented here exemplifies the results of the most recent FEAST/2 white-box system dynamics process modelling. The next step in this work is to ground the process dynamics in a detailed case study of an evolving software system and its process. Fitting and calibration of the model to a specific real process will help refine the simulation of a dynamics process by guiding clarification of variable interrelationships.

Validation (Forrester & Senge 1980) and application of the model to a real-world decision-making process needs, *inter alia* calibration, against data obtained from industrial software processes. Though the model, as presented here, has not yet been calibrated to industrial data, the behavioural patterns it generates (e.g., F05 in Figure 5d) are reminiscent of the long term growth rate decay observed in the systems studied in FEAST and modelled by means of an inverse square model (Turski 1996, FEAST 2000). Thus we deem it appropriate to offer the model in its present state as a significant step that includes the level of aggregation needed (considered essential to produce relevant models) and set of concepts needed to model and support long-term software evolution decision making.

Customisation of the model to a particular process may have to include further aspects not considered in the present model. These could include, for example, measures of user satisfaction and the impact of this on the dynamics. A particular factor cannot be achieved without a satisfactory definition that is probably domain dependent. A simple initial definition might, for example, be associated with some function of the difference between input and output work flows. This question requires further exploration and may involve the inclusion of a system *value* element, another concept already being considered elsewhere (Boehm & Sullivan 2000). Other aspect that may have to be considered includes the role of technology change as a driver of software evolution, as considered in a previous FEAST modelling exercises (Chatters *et al* 1999). Further aspect is concerned with the role of discrete vs continuous activities, and in particular, the role of field trial events (Lehman & Wernick 1998; Wernick & Lehman 1998) as opposed to the continuing validation activity assumed in the present model. Last, but not least, the consequences of different release policies (e.g., Woodside) may be an aspect that could be studied by means of the model presented here.

The variables defined in the model represent a set of evolution attributes from which a set of metrics can be derived. Such metrics will have a *systemic* character, analogous, to a certain extent, to the concept of *state* in control system theory. When applied to an industrial process, the latter will enable *monitoring* of productivity, process performance and the effect of process changes (e.g., improvements, new technology) at a high level of abstraction or aggregation. In this sense, the model complements metric definition and use in the context of black-box modelling of the process (FEAST 2000).

Note that *information hiding* (Parnas 1972), and other good evolutionary practices (Lehman 2000) may make a contribution to complexity growth control, though mostly through prevention rather than correction - a

© G Kahen, MM Lehman and J.F. Rami. All rights reserved.

good thing of course. However, even under the best of conditions and processes *E*-type system, functional growth *without* a degree of intrinsic system complexity growth does not appear to be possible. This observation follows from the fact that functional growth implies complexity growth, however defined, in the application itself and in 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 against it.

Further discussion of this issue requires, *inter alia*, a formal definition of complexity and is beyond the scope of the present study. The present model is built upon the reasonable 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 and permit complexity growth to be effectively ignored as a model parameter. This issue is hoped to be further clarified when the model is aligned to a real industrial process.

12 Final Remarks

Society relies increasingly on software at all levels, as demonstrated by the role of software in businesses and organisations. Management of evolution processes is becoming an ever more critical issue. Such processes are complex and dynamic but our knowledge to understand and manage them is limited. However, by application of the scientific method to their study, and in particular, by building models in a step-wise top-down procedure, one may provide useful tools and answers to management challenges in specific instances and also derive some more general, even generic, tools.

13 Acknowledgements

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

14 References - An * indicates that the reference has been reprinted in Lehman & Belady 1985.

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

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

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

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

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

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

Chatters B. W., Lehman M. M., Ramil J.F. & Wernick P. (1999) “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

Checkland, P. (1981) “Systems Thinking, Systems Practice”, London: J Wiley.

Checkland, P. & Scholes, J. (1990) “Soft Systems Methodology in Action”, London: J Wiley

Chong-Hok-Yuen C.K.S. (1981) "A Phenomenology of Program Maintenance and Evolution", PhD thesis, Imperial College, Department of Computing, 302 pp

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

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

FEAST (2000) “*F*eedback, *E*volution and *S*oftware *T*echnology”, FEAST/2 Project Web Site, http://www-dse.doc.ic.ac.uk/~mml/feast/

Forrester, J. W. (1961) “Industrial Dynamics”, Cambridge, Mass.: MIT Press.

Forrester, J. W. (1971) “Counterintuitive Behaviour of Social Systems”, Technology Review, Vol. 73, pp. 52-67

Forrester, J. W. & Senge, P. (1980) “Tests for Building Confidence in System Dynamics Models”, In System Dynamics, Legasto A. A. Jr., Forrester, J. W. and Lyneis, J. M. (eds.) TIMS Studies in the Management Sciences, Vol. 14. North Holland, New York, 1980, pp. 209 - 228

© G Kahen, MM Lehman and J.F. Rami. All rights reserved.

Kahen G., Lehman M. M., Ramil J. F. & Wernick P. D., “Dynamic Modelling in the Investigation of Policies for E-type Software Evolution”, ProSim 2000, International Workshop on Software Process Simulation and Modelling, 12 - 14 Jul. 2000, London, UK

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

Kleinmuntz, D. N. (1993) “Information Processing and Misperceptions of the Implications of Feedback in Dynamic Decision Making”, System Dynamics Review, Vol. 9, pp. 223-237.

Langley, P. A. (1998) “Using Cognitive Feedback to Improve Performance and Accelerate Learning in a Simulated Oil Industry”, WP-0027, LBS, UK.

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

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

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

Lehman, M. M. (1989) “Uncertainty in Computer Application and its control through the Engineering of Software”, Journal of Software Maintenance, Research and Practice, Vol. 1, pp. 3-27.

Lehman, M. M. (1994) “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 Software Technology, sp. is. on Software Maintenance, v. 38., n. 11, 1996, Elsevier, 1996, pp 681 – 686.

Lehman, M. M & Belady, L. A. (1985) “Program Evolution - Processes of Software Change”, Academic Press, London

Lehman M. M. & Wernick P. (1998), System Dynamics Models of Software Evolution Processes, Proc. Int. Wrkshp. on the Principles of Software Evolution IWPSE-98, ICSE-20, 20-21 Apr. 1998, Kyoto, Japan, pp. 6-10

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

Meadows, D.H. *et al* (1972), "Limits to Growth", Signet

Parnas, D.L. (1972), "On the Criteria to be Used in Decomposing Systems into Modules", CACM, Vol. 15(12), pp. 1053-8

Parnas, D.L. (1994), “Software Aging”, Proc. 16th ICSE, May 16-21, Sorrento, Italy, pp. 279-287

Royce W.W. (1970), "Managing the Development of Large Software Systems: Concepts and Techniques", Proc. WESCON, August 1970

Ruiz, M. & Ramos I. (2000) "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

Schach S.R. (1990), Software Engineering, Irwin and Aksen, Homewood, IL, 1990, p. 499

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

Sterman, J. (1994) “Learning in and about Complex Systems”, System Dynamics Review, Vol. 10, pp. 291-330.

Turski, W. M. (1996) “A Reference Model for the Smooth Growth of Software Systems”, IEEE. Trans. Softw. Eng., Vol.22, No. 8, pp. 599-600.

Vensim (1995) - Ventana Simulation Environment, Reference Manual, Version 1.62, Belmont, MA.

Wernick P. & Lehman M. M. (1998) "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, Vol. 46, Numbers 2/3, 15 April 1999

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

*Woodside, C.M. (1979), "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

Zurcher, F.W. & Randell, B. (1968), Iterative Multi-Level Modeling - A Methodology for Computer System Design, IBM Res. Div. Rep. RC-1938, Nov. 19678. Also in Proc. IFIP Congress 1968, Edinburgh, Aug 5 - 10, pp D-138 - 142

Appendix - The System Dynamic Model in Vensim language (Vensim 1995)

```
  Acceptance Flow =
  IF THEN ELSE((Work Accepted<ACCEPTED TARGET)
:AND:(Work Identified>0),
  300,0)
      ~
      ~     |

  ACCEPTED TARGET = 100
      ~
      ~     |

  Additions and Changes Identified by Others = (RANDOM
POISSON(60))
      ~ Changes/Month
      ~     |
  Anti regressive effort =IF THEN ELSE (
  (TEAM SIZE
  -Preparation effort
  -Progressive effort
  -Validation and Integration effort)>0,
```

© G Kahen, MM Lehman and J.F. Rami. All rights reserved.

```
(TEAM SIZE
-Preparation effort
-Progressive effort
-Validation and Integration effort),0)
    ~
    ~     |

Anti regressive work= Anti regressive effort * Productivity
    ~
    ~     |

Change plus defect discovery factor = 1/120
    ~
    ~     |

Cumulative Anti Regressive Work =INTEG(Anti regressive
work,0)
    ~
    ~     |

Cumulative Fielded Functionality = INTEG(Software
release,0)
    ~
    ~     |

Cumulative Progressive Work = INTEG(Implemented,0)
    ~
    ~     |

Demand obsolescense = Work Identified * 0.05
    ~ Changes/Month
    ~     |

F PROGRESSIVE FRACTION = 0.3
    ~
    ~     |

Fielded Functionality Satisfying Current Needs =
INTEG(Software release
-Requirement Change flow
,150)
    ~  [0,?]
    ~     |

Implemented =
IF THEN ELSE(In Progress > 0,
Progressive effort*Productivity,0)
    ~
    ~     |

In Progress = INTEG(Submision Flow-
Implemented+Rejected as needing rework,100)
    ~  [0,?]
    ~     |

IN PROGRESS TARGET =100
    ~
    ~     |

INTEGRATION PRODUCTIVITY FACTOR = 2
    ~
    ~     |

INTEGRATION SUCCESS FACTOR = 0.95
    ~
    ~     |

NORMAL PRODUCTIVITY = 2
    ~ Modules/Person Month
    ~     |

Preparation effort =Progressive effort *PREPARATION
EFFORT MULTIPLIER
    ~
    ~     |

PREPARATION EFFORT MULTIPLIER
= 0.5
    ~
    ~     |

Preparation Flow = Productivity*Preparation effort*
PREPARATION PRODUCTIVITY FACTOR
    ~
    ~     |

PREPARATION PRODUCTIVITY FACTOR = 2
    ~
    ~     |
 Productivity =NORMAL PRODUCTIVITY*
( (TEAM SIZE^0.2) - ( (1/1800) *TEAM SIZE^2) ) *
(1-MAX(0,
SYSTEM TYPE MULTIPLIER*
(Cumulative Progressive Work - Cumulative Anti Regressive
Work) ))
    ~ Modules/Person Month
    ~     |

Progressive effort = F PROGRESSIVE FRACTION*
TEAM SIZE
    ~
    ~     |

Rejected as needing rework =
Productivity*Validation and Integration effort*
(1-INTEGRATION SUCCESS FACTOR)*INTEGRATION
PRODUCTIVITY FACTOR
    ~
    ~     |

RELEASE POLICY ([(0,0)-(100,10)],(0,0),
(11,0),(12,1),(14,1),(15,0),
(23,0),(24,1),(26,1),(27,0),
(35,0),(36,1),(38,1),(39,0),
(47,0),(48,1),(50,1),(51,0),
(59,0),(60,1),(62,1),(63,0),
(71,0),(72,1),(74,1),(75,0),
(83,0),(84,1),(86,1),(87,0),
(95,0),(96,1),(98,1),(99,0),
(107,0),(108,1),(110,1),(111,0),
(119,0),(120,1),(122,1),(123,0),
(131,0),(132,1),(134,1),(135,0),
(143,0),(144,1),(146,1),(147,0))
    ~
    ~     |

Requirement Change flow =Fielded Functionality Satisfying
Current Needs*
Change plus defect discovery factor
    ~ Changes/Month
    ~     |

Software release =MAX(0,(Work Ready to Release/TIME
STEP)*
LOOKUP EXTRAPOLATE(RELEASE POLICY, Time))
    ~
    ~     |

Submision Flow = IF THEN ELSE((In Progress<IN
PROGRESS TARGET)
:AND:(Work Prepared for Implementation > 0),300,0)
    ~
    ~     |

Successfully integrated =
 IF THEN ELSE
(Work Implemented > 0,
Validation and Integration effort*Productivity* INTEGRATION
SUCCESS FACTOR * INTEGRATION PRODUCTIVITY
FACTOR
,0)
```

© G Kahen, MM Lehman and J.F. Rami. All rights reserved.

```
    ~
    ~     |
SYSTEM TYPE MULTIPLIER =0.0005
    ~
    ~     |
TEAM SIZE = 30
    ~
    ~     |
TIME STEP = 0.125
    ~
    ~     |
Validation and Integration effort =
Progressive effort * VALIDATION AND INTEGRATION
EFFORT MULTIPLIER
    ~
    ~     |
VALIDATION AND INTEGRATION EFFORT MULTIPLIER =
0.5
    ~
    ~     |
Work Accepted = INTEG(Acceptance Flow-Preparation
Flow,100)
    ~
    ~     |
Work Identified=INTEG(Additions and Changes Identified by
Others+
Requirement Change flow
-Demand obsolescense-Acceptance Flow ,600)
    ~ Changes
    ~     |
Work Implemented = INTEG(Implemented-
Successfully integrated-
Rejected as needing rework,200)
    ~ Changes
    ~     |
Work Prepared for Implementation = INTEG(Preparation
Flow-Submision Flow,100)
    ~
    ~     |

Work Ready to Release = INTEG(Successfully integrated-
Software release,0)
    ~  [0,?]
    ~     |
********************************************************
    .Control
********************************************************~
Simulation Control Paramaters
    |
FINAL TIME  = 150
    ~ Month
    ~ The final time for the simulation.
    |
INITIAL TIME  = 0
    ~ Month
    ~ The initial time for the simulation.
    |
SAVEPER  = 1
    ~ Month
    ~ The frequency with which output isstored.|
```

© G Kahen, MM Lehman and J.F. Rami. All rights reserved.