Software Process White Box Modelling for FEAST/1

P Wernick and MM Lehman

Department of Computing
Imperial College
180 Queen's Gate
London SW7 2BZ
England

tel.: +44 (0) 171-594 {8216 8214}
fax: +44 (0) 171-581 8024

{pdw1 mml}@doc.ic.ac.uk

Abstract

This paper describes a high-level system dynamics model of a real-world software evolution process. This process is implementing embedded software elements of a defence system composed of hardware and software components. The model is one of the outputs of the FEAST/1 project, which is investigating the role and effect of feedback in the global software process. The simple feedback-based model, which has resulted from a top-down modelling approach, demonstrates the influence of the global process on the evolution of the software specification and implementation. Model outputs closely simulate actual and expected metrics for the real-world project. It is concluded *inter alia* that feedback external to a software production process may significantly influence that process.

Keywords

Laws of software evolution, system dynamics, software process, modeling, simulation, feedback, FEAST, E-type systems

1. Background and Context

As part of a continuing investigation into the role and impact of feedback on the long-term evolution trends of software systems, models of several current real-world software processes are being built. The current FEAST/1 investigation (Lehman and Stenning, 1996) is intended initially to find support for, and later to examine the implications of, the VIIIth Law of Software Evolution (Lehman and Ramil, 1999). This states that:

> *'E-type evolution processes constitute multi-level, multi-loop, multi-agent feedback systems and must, in general, be treated as such to achieve significant process improvement for other than the most primitive processes.'*

System Dynamics (Forrester, 1961) (SD) is being used to explore the dynamics of the long-term software process, with the Vensim environment being employed to construct and run executable models. The objective of this work is to examine the evolution over many software releases of a number of real-world software products, each in its environment of developers, managers, field testers, vendors, users, and others.

The SD model described here has been developed to simulate the evolutionary influences arising during the development of a new software product forming part of a larger, also new, hardware and software system. The time covered in the simulation runs from the start of software

development to the completion of a final pre-production prototype of the complete system. The model focuses on the evolution of the software specification during the software system's development, and the convergence of the implementation towards that specification. This evolution reflects the clarification of possible and/or desirable product performance and other attributes as knowledge and understanding increase.

Results from parallel 'black box' studies (Lehman *et al.*, 1997) are being used *inter alia* to provide data for testing and validating these models. By considering product metrics taken from a sequence of software releases as abstract numeric indicators, the black box studies are seeking evidence of feedback-related software evolution phenomena and of consistency with phenomena previously observed in other systems and encapsulated in Lehman's Laws of Software Evolution (Lehman *et al.*, 1997, 1998). The aim of these studies is the building of interpretable mathematical models replicating actual observed behaviour (Turski, 1996). The SD models are providing suggestions both for new analytical strategies and for new metrics to examine in current and future black box studies.

For this software evolution study, the process examined is viewed as a long-term multi-release software evolution process, with total system integrators and external trials teams being the customers for releases of the software product in the world outside the software development environment. The information received by the software developers from trials is seen here as feedback from outside the software production process.

2. The Process Described

2.1. The Software Process and its Context

The industrial process whose software element is considered here is typical of those employed for the development of defence systems and their support equipment by the organisation concerned. In general, such systems comprise both hardware and software components. The developments are each undertaken for a specific customer under a development contract usually lasting several years. In these contracts, the price and timescales are both initially fixed in relation to an agreed, predefined, initial product specification. As noted previously, these will inevitably lead to changes in specification due to learning during the development process. The costs of these changes are generally absorbed by the developers. The contract price and/or timescale may, however, be modified if warranted by, for example, development progress, trial results or changes in the customer's requirements or operational domain.

The software development process includes decomposition of the specification into functional areas, design of an implementation using both known and innovative mechanisms, implementation and unit testing of that design, and integration testing of the implementation in its hardware environment in test rigs designed to exercise both software and hardware components. The culmination of each design, implementation and testing cycle is the preparation of a software release which is integrated with hardware elements to form a prototype of the complete system. This prototype is then subjected to a *field trial*, which tests the behaviour of the some or all of the complete system against parameters set in the specification and/or the development contract. The information fed back to the software developers from each field trial includes reports of changes needed in the specification, and consequently in the implementation, in order to meet the product integrators' or customer's view, changed to reflect the trial's results, of the final system.

The minimum functional content of the software to be released for integration and testing in each

trial, and the trials timetable which sets the time allowed to develop each software release, are outside the control of the software development managers. In the process being modelled, implementation of approximately 95% of the original specification was needed by the time of the first field trial to provide sufficient functionality for that trial to proceed. The first trial occurred after about 30% of the contracted project time. The software development work has therefore been heavily front-end loaded. As the project progresses, field trials become more frequent, and software development cycles shorter.

Progress payments of commercially significant amounts are made to the system's developers at predetermined milestones during the contract time, conditional on successful field trial completion. The over-riding commercial imperative for software development managers is therefore to complete the functions required for a trial in agreement with the current specification by the pre-trial software completion deadline. According to these managers the consequence of this pressure is that under all but the most extreme circumstances the software development required for a trial is completed by the deadline, regardless of the number of staff-hours required. To ensure that completion occurs, overtime is usually worked in the period leading up to a field trial. If the resources available are still insufficient to meet the deadline, additional staff with suitable skills are employed as required on short-term contracts.

The schedules of software development and trials requirements are not perfectly aligned. Work on successive software releases tends to overlap, due to the implementation of some functions requiring more time than one release and field trial cycle to complete. Work on some functions is therefore expected to span more than one cycle. Also, functions ready before their scheduled delivery dates may be included in a trial in addition to the functionality scheduled for that trial, in order to obtain earlier feedback from the trials process than would otherwise be the case.

2.2. The Process and other Software Evolution Processes

The software evolution process described above differs in a number of respects from other processes being studied for FEAST/1, and from commercial software evolution processes in general. What is represented here is the *ab initio* evolution of a new system, including preliminary releases of development versions for external testing in predefined field trials. This may be contrasted with the development of new software functions as an evolutionary response to feedback generated by software product releases fielded to many users for use in a number of environments.

These differences, which are not directly reflected in the model described here, but which have been made more apparent by the construction of the model, specifically include:

- the project having only one customer. This simplifies the feedback mechanisms affecting the product's evolution;
- product performance and functionality being tested by a single sequence of field trials, resulting in a single stream of feedback information from outside the software production process. This contrasts with the multi-stranded feedback paths of a product whose many customers use it simultaneously. This single sequence eliminates the need for software developers to co-ordinate, reconcile, prioritise and react to numbers of feedback messages provided asynchronously by many users;
- progress payments, depending on successful field trial completion. These trials are therefore more financially significant in funding product development than is the case in typical

commercial software development contexts;

- a fixed project timescale, with penalties for later delivery larger than those normal for commercial fixed-price contracts; and
- a greater degree of risk in the project than may be typical for commercial projects, due to the leading-edge nature of the development work.

These differences have the consequence that processes described here, the issues affecting (and affected by) long-term software product evolution, and models intended to represent those processes and issues, may differ from those relevant to a commercially-distributed software product. However, despite these differences the process described here may bear similarities to commercial software processes during which iterated user-based prototyping is used to learn and/or to check users' and developers' understanding of the problem and the feasibility of proposed solutions.

3. The Model Described

3.1. Model Features

The main structure of the model is illustrated in Figure 1. It simulates the software production process, the field trialling of the software implementation as integrated into complete system prototypes, and changes made to the software specification as a result of feedback from the production process and from field trials.

As part of the model-building process the software specification has been abstracted into a number of arbitrary-sized 'units'. Each of these can be considered as the specification of some attribute(s) of the software product. The simulated implementation is also measured in these units, each of which represents in this case the coded, tested and documented implementation of one unit of specification. The process of implementation and initial testing is represented by a single time delay function,[1] which moves these units continuously from a stock of units awaiting implementation to one of units which are believed to have been implemented successfully, but have not yet been field trialled. This process is successfully completed for most units. Others are scheduled to take more than one cycle to complete, as noted above, or are delayed beyond their initially planned completion date due to their taking more time to implement than expected. Finally, a small proportion of units is found not to be implementable as originally or currently specified. These are eliminated from the specification, but may be replaced in the specification by new or changed equivalents. The rate of completion of successfully implemented units is represented in the model by multiplying the rate at which work on specification units is completed (whatever the outcome) by a 'success percentage' parameter which remains constant throughout the simulated project.

A further source of changes to the specification at this stage is that work leading to the successful implementation of a specification unit may itself result in the identification of changes or additions to the specification. The flow through this feedback path is also represented by a

---

[1] The delay used is a *third order* delay, a built-in function of SD environments, which is often used to represent the time delays caused by some entity passing through the (abstracted) stages of a process made up of a *sequence* of sub-processes, each of which depends for its input on the output of the previous sub-process. This description fits the technical software process. The same delay function has been used in other FEAST/1 software process SD models to represent the technical software process (Chatters *et al.*, 1998)

constant proportion of the successful implementation flow. It will be observed that feedback within the software process is present, and affecting the specification, before the first field trial occurs.

Following the first field trial, additional feedback paths come into play. The information from each field trial may result in a need to re-examine and possibly modify parts of the specification. Elements of the latter may be found to be wrong or unnecessary, and must be removed from the system. New elements may need to be added to the specification. Each of these drivers of specification change is reflected in information feedback paths in the model, again calculated as constant proportions of the successful implementation flow.

The current model represents the sequence of field trials after the first as a continuous process, abstracting out the actual trial dates. The reduction in interval between trials as the project progresses and the consequent shortening of software development lead times is simulated by a progressive reduction in the time delays in the feedback paths of trial results to software developers, and in the time allowed for implementation work to be started and completed on a unit of specification.

The model also incorporates an ability to reflect exogenous changes made in the specification during the project. These structures are not shown in Figure 1.

3.2. Simplifications in the Model

The emphasis in FEAST/1 model building has been on simplicity. The objective has been to achieve an initial model which reflects that aspect of the development process behaviour under investigation. Such an approach will make it easier to use the models to initiate the process of identifying feedback patterns, determine their impact on the software production process and its products. It will also facilitate the process of model calibration and validation. Given a successful first step, the model can then be refined as required.

The underlying approach to model construction has therefore been top down. The initial model presented here reflects a very high level view of the process and high level process behaviours. Future plans include the refinement of the model to enable it to simulate more detailed aspects of real-world behaviour (see Section 8 below).

This approach has resulted in the following features of the real-world process being simplified or abstracted in the initial model described here:

- resource: Resource provision, management and limitations have been ignored as factors affecting the progress of the project since, as noted in Section 2.1 above, resources are made available as required to meet project deadlines;
- pre-trial testing: The effects of pre-field trial testing procedures, in terms of the resulting information fed back to the software developers and any necessary rework or additional work, are abstracted within the calculation of the flows of specification units successfully implemented ready for field trial, of new units arising from that successful implementation, and of units either delayed to later field trial cycles, or abandoned and replaced;
- continuous development: As previously stated, all development after the first field trial is represented by a single continuous activity spread over the time remaining to complete the contract. This continuous mechanism abstracts discrete timings of individual events such as individual field trials, the generation of feedback from those trials and its use by the software development team;

- the learning process: As a first approximation, the current model makes no provision for efficiency gains during the process as software developers learn about the process or product;
- bug fixing: It is assumed that the resources provided to complete software development on time include provision for the fixing of implementation faults. It is further assumed that implementation fault-related activities are not allowed significantly to slow down overall progress towards completion. Implementation fault rework mechanisms have therefore been excluded from the model. The feedback path designated in the model as 'rework' therefore refers only to the information flow needed to correct faults in the *specification*;
- fixed final timescale: Though project timescales may be renegotiated during the project time (see Section 2.1 above), the model's calculations are predicated on the initially fixed completion time for the project remaining unchanged. This is a reasonable simplification when no significant changes are demanded by the customer during the project's progress; and
- process context: The model describes a software process which is itself only a part of a much larger software and hardware product development process. These higher level aspects have been abstracted in the model described here. A model of this software process which includes aspects of the wider environment has been previously reported (Lehman and Wernick, 1998).

4. Results Obtained from the Model

Good agreement between the behaviour of outputs under varying simulated conditions of a model, and that of the real-world system which the model seeks to represent, increases the probability that the model will provide a sound basis for investigations. For the long-term goals of the work reported here, these investigations are to include examinations of predicted changes in behaviour of the real-world process in pursuit of global feedback process optimisations, and will be performed by changing model parameters and/or structure. After successful model calibration, such investigations can be performed with greater confidence in their predicted outcomes. The process of calibration of itself also provides increased understanding of the modelled process.

For all of these reasons it has been essential to calibrate the model presented here against actual or estimated data. The following subsections describe the means by which input parameters needed to calibrate the model have been obtained, and quantitative and qualitative[2] results of simulations exercises.

4.1. Calibrating the Model Inputs

In order to support model-based simulation exercises, it is necessary to obtain accurate, or at least realistic, values for the model's input parameters. The process under examination here is currently a little over half-way through the expected project time, and little hard data has been collected for model parameters such as time delays, implementation completion success rates and trial feedback factors. Values for these and other parameters have therefore been obtained primarily from estimates provided by collaborator experts for this and similar projects within the same organisation. These experience-based quantitative estimates are being used to provide initial assurance that the model is behaving in a reasonable fashion. As more actual data become

[2] i.e. trends over time in model outputs and real-world process or product metrics, without exact quantitative agreement

available for the project, and as experience grows in collecting this data, more accurate values taken from actual experience are expected to replace some or all of the experts' estimates.

This informal approach to model calibration has proved particularly effective in the industrial environment being examined, since it has been possible to obtain useful results from the model cheaply and quickly. A more detailed approach to the problem of calibration might have failed in such an environment due to the need to expend time and resource before the model could demonstrate its usefulness.

4.2. Comparing Models Outputs with Real-World Values

Figures 2 and 3 show respectively model outputs simulating the change in size of the product's specification and implementation over the project, and the effort employed over the project. For the purposes of presentation, results are shown for a fictional project with values of 1000 specification units for the initial specification, and 100 time units for the contract time.

For the same reasons as described for input parameter calibration, actual data from which to distil reference modes[3] against which to check the model are incomplete or unavailable. It has for instance not proved possible to obtain actual figures for changes in the size of the software specification, since the documentation, in its current form, is not amenable to such an analysis. Reference modes for development progress and growth in specification size over the project have therefore been generated on the basis of project managers' and expert developers' judgement, their analogies with previous similar projects, and their expectations of the current project. Data is available for some of the project time on the effort expended on software development, measured in person-hours. This has been used, in normalised form, for the calibration of the simulated output for effort expended.

4.2.1. The Specification and its Implementation

Model outputs reflecting rising trend in the size in specification units of the implemented software as the system grows towards its final implementation, and the slight growth in that specification over the project from its initial value, are shown in Figure 2. The model output simulating the current size of the implementation shows fast growth until the time of the first field trial. After this trial, there is a slight dip in implemented system size as some of the specification as originally implemented is invalidated by field experience. This is followed by a slower rate of growth in the size of the successfully trialled implementation, gradually converging with the size of the specification. By the conclusion of the contract, the flow of information feedback of learning from field trials will have died away and the specification and implementation will be of the same size, reflecting the need for the implementation to be an accurate reflection of the specification at the end of the project. The plot also reflects the expected slight net growth in the specification itself over the project due to learning during implementation and trials. This increase is however too small to be observable in Figure 2.

The trends displayed by the model in both specification growth and the rate of implementation are in accordance with the expectations of the collaborator's experts for projects of this type within their organisation, although they may not be generalisable to other software development product types, organisations or contexts.

---

[3] This is a pattern of expected or actual behaviour of a modelled output over time, used to check the operation and dynamic behaviour of SD models.

4.2.2. Effort Expended

Figure 3 shows the simulated value for effort expended in developing the software (the smooth line) plotted against the actual trend in effort for the project to date (the jagged line, indicating a moving average of normalised values of person-hours charged).

The calculation of simulated effort is based on the collaborator experts' view that new specification units generated during the development process require more effort to implement than elements of the original specification. This increased effort per unit is due to aspects such as the replacement by more complex solutions of simpler original assumptions shown to be unfeasible during the development process. It also reflects the growing difficulty in this environment of fitting new functions into the existing implementation. As work progresses, increasing areas of the design have been successfully trialled and signed off, resulting in significant resistance to changes which would ease the task of implementing new functions. Units which require more than one cycle to complete are also assumed to be more difficult to implement than those elements of the original specification which can be implemented in one cycle, due both to their inherent complexity and to the fact that the system specification which they assume to be static may change as a result of a field trial occurring during the implementation process.

The simulated value shown in Figure 3 reflects all of these elements of increased difficulty, and therefore cost, by weighting the effort calculation for generated and each cycle of multi-cycle work according to the square of the current size of the implementation. This derives from the belief that the growth in the *complexity* of the existing implementation, which as stated is a factor in making the addition of the next increment of implementation more difficult, is reflected by changes in the *square* of the size of that implementation. The calculation also recreates the expected trend of continually increasing difficulty over the project timescale in implementing each newly-started specification unit in the context of an increasing body of successfully trialled, and therefore effectively unchangeable, code.

The average of the weighting factor used for Figure 3 is close to project experts' estimates of the average relative implementation costs of multi-cycle or generated specification units against initial specification units implemented over a single cycle. Since the appropriate information is not available, it has not been possible to calibrate the model to reflect actual numeric values. Nevertheless, the model demonstrates high-level dynamic behaviour very close to that of the (normalised) actual data.

5. Sensitivity Analysis

The model described above has been subjected to an initial sensitivity analysis to determine whether it exhibits realistic behaviour under anomalous conditions, and to identify model parameters or structures which have a major effect on model behaviour and thus possibly on real-world project behaviour. The effects on simulation outputs of changing individual input parameters have been shown to correspond to what would be expected from equivalent changes in actual projects. This demonstrates that, from a phenomenological viewpoint, the model is realistic.

An extreme example of this is provided by increasing the feedback additional work generation factors until they are greater than one. This results in a simulated pattern of specification unit completion against new unit generation which, instead of converging to a point at which all of the specified work has been completed and trialled, diverges increasingly, and eventually

exponentially, as the end of the project approaches. This is due to the increasing inability of the implementation and trialling procedures to keep pace with the generation of new work, since that new work is being generated at a faster rate than existing work is (or can be) done. Such a divergence would therefore be expected for a real-world project under these circumstances, resulting in a project which can never be successfully completed but which continues to consume resource at an ever-growing rate whilst growing ever further from its goal of a completely implemented specification. This situation parallels the instabilities previously found in OS/360 evolution after Release 20 (Lehman and Belady, 1985). The collaborator experts' response to this model behaviour was that they had known of a real-world project which had behaved in this fashion, and that their advice had been to cancel the project immediately since no successful outcome was foreseeable.[4]

6. Conclusions and Implications of the Model Building and Simulation Process

Conclusions and implications related to the software process which have been drawn so far from developing and calibrating the model include the following:

- a *simple* model of the software process can simulate behaviour patterns of the real-world process. This simple model has also successfully reproduced situations under which problems arise with a project, such as the behaviour identified in Section 5 above. Together these suggest that the model structure may reflect real-world high-level processes rather than merely comprising an abstract mechanism which produces 'correct' outputs;
- the ability of a *feedback-based* model to reproduce real-world behaviour trends lends support both to the hypothesis that feedback is an important influence on projects such as that examined here, and more generally to the VIIIth Law of Software Evolution (Lehman and Ramil, 1999);
- real-world trends in software evolution can be simulated by a model which abstracts some activities and aspects of great importance to the long-term survival of a software product. In the example presented here, these abstracted activities and aspects include the processes and costs of implementation fault identification and correction, and the peaks arising in required work rate as release deadlines approach. It should, however, be noted that the change in inter-release interval over time is significant in determining this model's dynamic behaviour. The successful calibration of a model which abstracts these activities and aspects also suggests that they may have only second-order effects on project trends, since they appear not to constrain the high-level project specification and implementation trajectories;
- the successful calibration of the model suggests that, at this level of abstraction, resources will be provided to complete implementation work if the demand for this work, or the cost of not having them, is sufficient to support a business case for the expenditure concerned. Resource provision can thus be seen in this instance largely as a direct *effect* rather than as a *cause* of evolutionary pressures. This viewpoint is supported by the collaborator;
- this model, calibrated with constant input parameters, demonstrates reasonable simulation behaviours, implying either that these parameters are constant over the project or that changes in them during the project may have little impact on high-level specification evolution trends. This further suggests that stable dynamics for the project can be established

[4] It should be noted that the project under investigation is not suffering like this, and the example of such a failure noted by the expert was not in a project being undertaken by the FEAST/1 collaborator.

at a very early stage, a conclusion supported by FEAST/1 statistical studies (Lehman *et al.*, 1997), and may allow future project progress to be estimated from early results; and

- whilst the stress in the current FEAST work has been towards process improvement through the management of the feedback process, the dynamic behaviour identified may also explain the success of cost estimation and planning tools such as COCOMO (Boehm, 1981; COCOMO II, 1998). Once a project and its organisational context have been established, their dynamics are likely to be sufficient to determine project characteristics such as duration and cost.

In addition, the need in building this model to make explicit the differences between the type of software process described here and the evolution process of a commercial software product (see Section 2.2 above) has also raised the question of whether, and if so at what level of abstraction, all software evolution environments can be considered to be equivalent. It exemplifies the fact that not all software evolution environments are equivalent at the detailed level. It may be asked whether there exists some higher level of abstraction corresponding to a baseline at which these environments are all equivalent. This issue is to be explored in more depth in follow-on projects, by extracting common model concepts, structures and elements from models built for the current project's industrial collaborators (Lehman, 1998).

Looking to a wider context than the software process, since the software production process in this successfully calibrated model has been reduced to a single delay function of a type used in SD models to abstract a variety of production processes and human activity systems, a model of the global product evolution process has been developed whose structure may not be confined to software processes. At the level of abstraction of this model, similarities between software production processes and other processes may be greater than is sometimes claimed, and models of the type described here may well assist in the identification and exploitation of these similarities. It also suggests that the software production process may act in a manner very similar to that of other human activity systems which are constrained by similar external feedback systems.

7. Related Work

The SD model of the software process presented here differs significantly from those of others researchers. These differences have arisen from a desire to understand the software production process within its wider context, resulting in a concentration on the high-level feedback elements surrounding the software production process and the deliberate abstraction of that process. The approach underlying the this model can be contrasted with that of those who have modelled large areas of the internal structure of the software production process, such as Abdel-Hamid and Madnick (1991), or specific aspects of that process, as exemplified by Madachy's (1996) work on inspections.

Powell and Mander (1999) also consider the internals of the software production process, in order to examine concurrent development within a cycle and optimise the total cycle time and process stage concurrency. A reduction in development cycle time would be reflected in the model presented here only by a shorter implementation delay, since the simple calculation of the delay in this model abstracts the very aspects of the process with which Powell and Mander are concerned. At the next higher level of abstraction, the concurrent development of software elements is implicit in this model.

Williford and Chang (1999) take a similar long-term, high level view of the software process as

that taken here. Their objectives are also similar, in that they seek to optimise their corporate response to the demands of the global software process. However, their focus on the prediction of future staffing requirements has necessitated their adding to their process simulation details of resource aspects which have been abstracted in this work.

8. Next Steps

The model presented here forms the first step in a long-term, more widely-based examination of the role and impact of feedback in the global software process. It is therefore seen as a tool to be used in that examination. As a result, future work planned on the model will not be restricted to refining the current model but includes extending it to help meet the wider research agenda. In particular, it is intended that, at a later stage, this model, or its results in terms of deeper understanding of the long-term software process be incorporated into more broadly-based SD models of the global software process (Lehman and Wernick, 1998).

To enable the model to simulate detailed project behaviour more accurately and to provide better understanding of issues currently implicit in the model, investigations will be undertaken into the causal mechanisms affecting model elements identified in extended sensitivity analyses as having significant impact on model output trends. By directing future work on the model towards adding detail in parts of the model which appear to be most influential in the software evolution process, this will also provide a basis for identifying likely areas for potential real-world improvements in the process in general and improved feedback control in particular.

The scope of the current model will also be extended in exercises intended to calibrate it for other projects within the same organisation by changing only the input parameters. The objective of this work will be to determine whether such models encapsulate regularities or patterns of organisational behaviour across projects. Similar work will be undertaken for equivalent projects in other organisations, to ascertain whether the model also reflects regularities beyond those of one development organisation. In addition, consideration will be given to the potential generic applicability of this model to user prototyping-based commercial software processes.

Consideration of the wider research issues raised by the development and calibration of the model may also lead to investigations into issues such as the nature and effects of any identified underlying differences between defence-related research-oriented projects, as modelled here, and commercial long-term software evolution processes.

9. Summary

The model presented here has been developed as part of an examination of the effects of the surrounding feedback system on trends in a software process. It demonstrates that a very simple model, with few feedback paths and with the software production process reduced to a single step, is capable of simulating high-level behaviours of the global process. This suggests that influences outside that production process have an important effect on process and product trends, and that these influences must be taken into account when seeking to improve that process.

10. Acknowledgements

Gratitude is expressed to the FEAST/1 industrial collaborators (Matra BAe Dynamics, ICL, Logica and MoD DERA), and particularly to Mike Atterton, Les Barker, Bob Born, Paul Gilbert, Dave Nuttall and Rob White of Matra BAe Dynamics for help in developing the model described here. Thanks are also due to the other FEAST/1 team members: Dewayne Perry, Juan Ramil and

Wlad Turski, and further thanks to Juan Ramil for commenting on an early draft of this paper. Since October 1996 this work has been supported by EPSRC grants GR/K86008, GR/L07437 and GR/L96561.

References

Abdel-Hamid T. and Madnick S.E., *Software Project Dynamics – An Integrated Approach*, Prentice-Hall, Englewood Cliffs, NJ (1991)

Boehm B.W., *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, NJ (1981)

COCOMO II, home page, on-line at <http://sunset.usc.edu/COCOMOII/cocomo.html> (August 1998)

Chatters B.W., Lehman M.M., Ramil J.F. and Wernick P., *Modelling A Software Evolution Process, submitted to* 21st International Conference on Software Engineering (ICSE '99), Los Angeles, May 1999 (1998)

Forrester J.W., *Industrial Dynamics*, Productivity Press, Cambridge, MA (1961)

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

Lehman M.M. and Stenning V, *FEAST/1: Case for Support*, ICSTM – DoC EPSRC Proposal (March 1996)

Lehman M.M., Perry D.E., Ramil J.F., Turski W.M. and Wernick P., *Metrics and Laws of Software Evolution - The Nineties View*, Proc. Metrics '97, Albuquerque, NM, 5–7 Nov. 1997 (1997)

Lehman, M.M. *FEAST/2: Case for Support : Part 2*. ICSTM – DoC EPSRC Proposal (July 1998)

Lehman M.M., Perry DE. and Ramil J.F., *On Evidence Supporting the FEAST Hypothesis and the Laws of Software Evolution*, Proc. Metrics '98, Bethesda, Maryland, Nov. 20-21, 1998 (1998)

Lehman M.M. and Wernick P., *System Dynamics Models of Software Evolution Processes*, Proc. International Workshop on the Principles of Software Evolution, ICSE 20, Kyoto, Japan, April 1998, 6–10 (1998)

Lehman M.M. and Ramil J.F., *The Impact of Feedback in the Global Software Process*, keynote address, ProSim '98, International Workshop on Software Process Simulation Modeling, June 22–24, 1998, Silver Falls, Oregon, USA; *to appear in Journal of Systems and Software* (1999)

Madachy R.J., *System Dynamics Modeling of an Inspection-Based Process*, Proc. ICSE 18, IEEE, 376–386 (1996)

Powell A and Mander K., *Strategies for Lifecycle Concurrency and Iteration – a Systems Dynamics Approach*, *presented at* ProSim '98, International Workshop on Software Process Simulation Modeling, June 22–24, 1998, Silver Falls, Oregon, USA; *to appear in Journal of Systems and Software* (1999)

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

Williford J. and Chang A., *Modeling FedEx's IT Division: A System Dynamics Approach to Strategic IT Planning*, *presented at* ProSim '98, International Workshop on Software Process Simulation Modeling, June 22–24, 1998, Silver Falls, Oregon, USA; *to appear in Journal of Systems and Software* (1999)

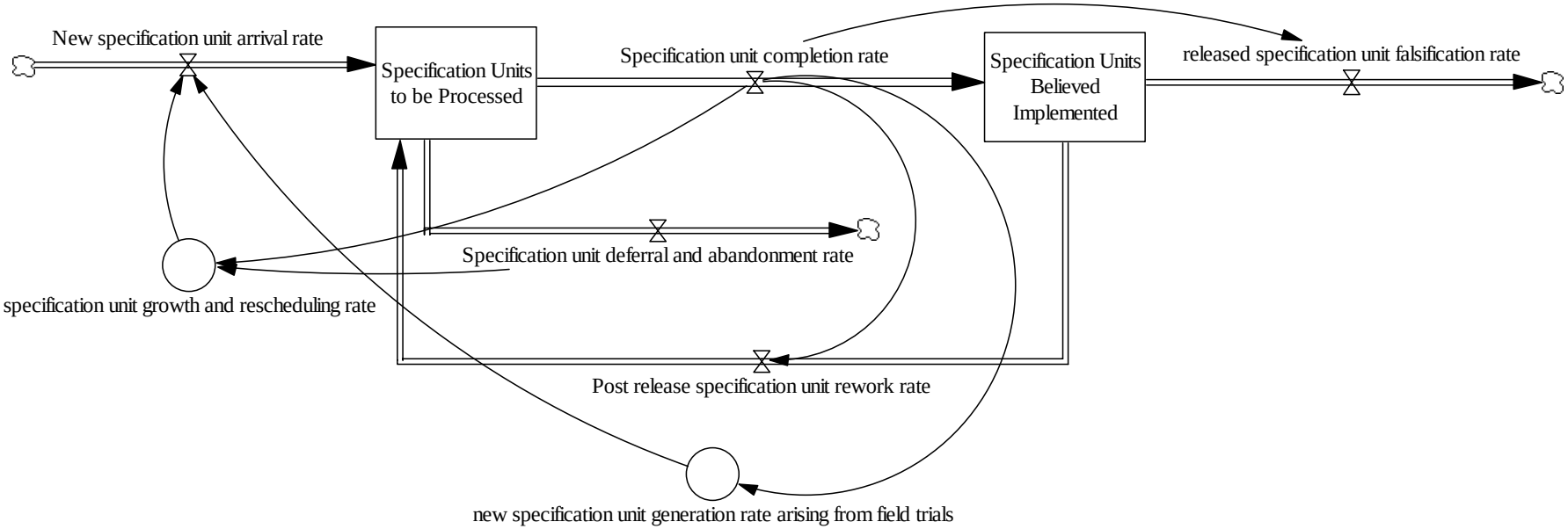

Figure 1 : Outline of the Model Structure

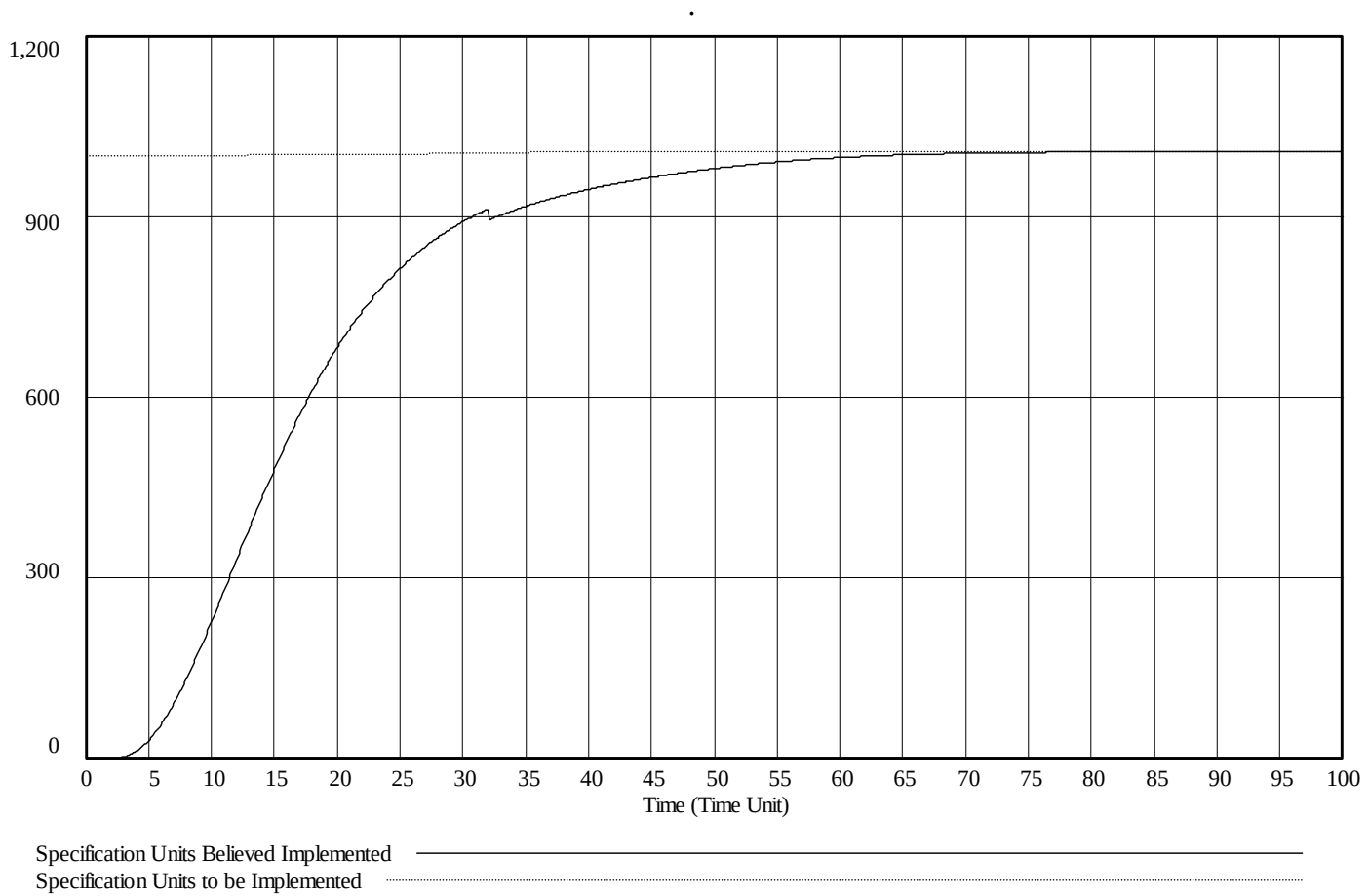

Figure 2 : Specification and Implementation Growth Over Project

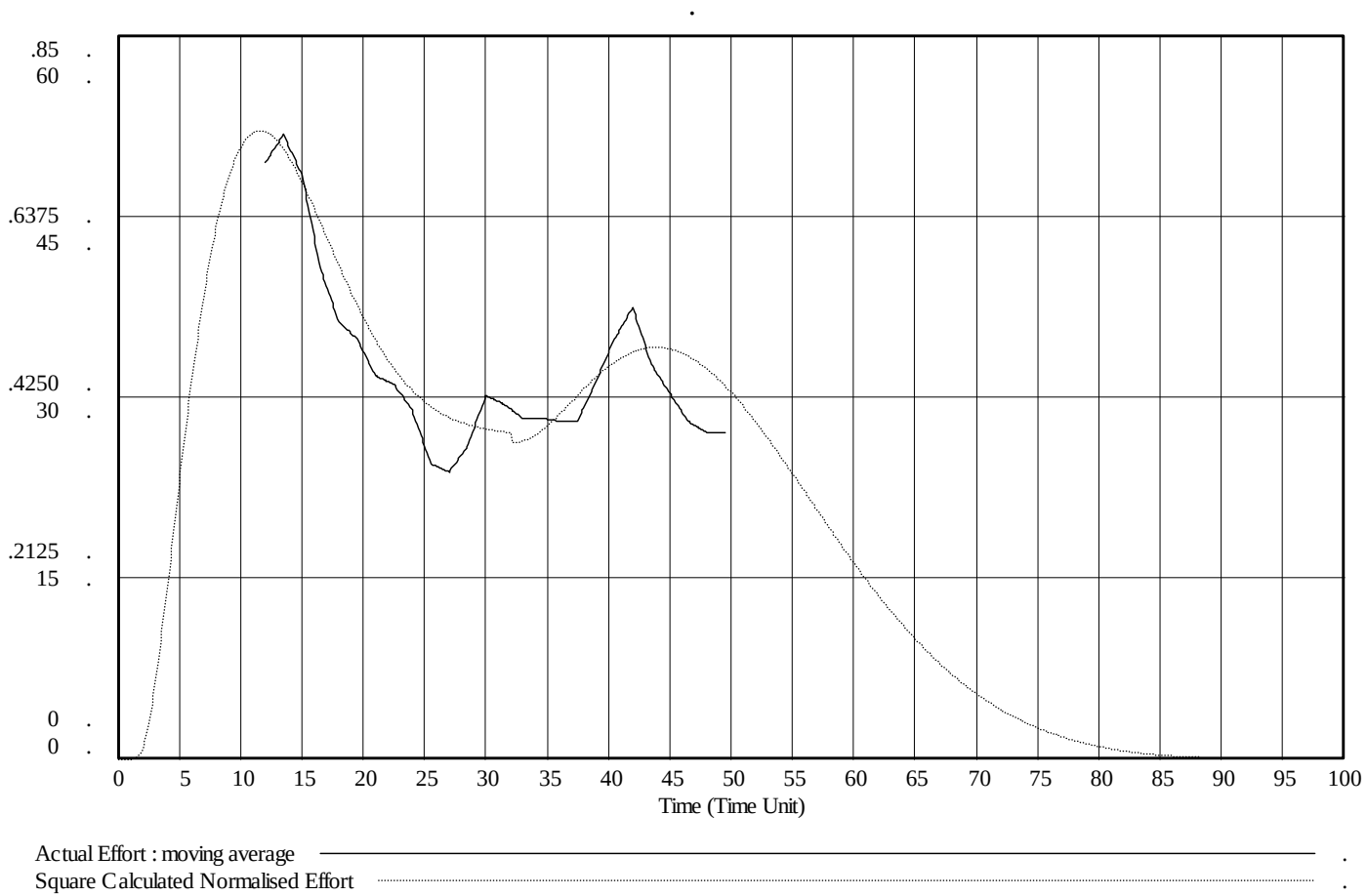

Figure 3 : Effort Expended in Project (Square of size based factor)