Adding Learning to Software Process Models

Paul Wernick

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

pdw1@doc.ic.ac.uk

Department Technical Report DOC 97/13

ABSTRACT

This paper considers an issue raised by Lehman at ISPW 9, where he stated that the software process is a 'learning' process. We examine the ramifications of that statement, and of the nature and impact of that learning, in the context of the process controlling the evolution of a software product over time and multiple releases.

We present a high-level model of the software process which describes that process in terms of the gaining and storing of knowledge of a software product and its use and environment, and of the application of that knowledge in making changes to the product.

1. INTRODUCTION

This paper considers an issue raised by Lehman at ISPW 9 [mml94], where he states (p.136, numbered point #1) that the software process is a *learning* process. We examine the ramifications of that statement, and of the nature and impact of that learning, in the context of the process controlling the evolution of a software product over time and multiple releases.

Some initial thought has been given to this question during initial investigations for the white box software process modelling effort for the FEAST/1 [mml96] project, examining feedback-related issues in the software process in general, and the place of learning in that process in particular, has caused a number of questions to arise concerning Lehman's statement. These questions are:

- what does 'learning' mean in this context?
- what is being learned?
- why is learning considered to be inevitable?
- how is the learning spread amongst those who need to know it?
- how is that learning applied – how does the achievement of that learning affect the world?
- what effect does the *level* of what is learned have on the product and its evolution?

These questions are of immediate practical relevance, since the formulation of answers to them, and the design of models of the organisational systems in which learning flows, is stored, and is used, may need to be taken into account during the FEAST investigation into the feedback mechanisms in the software process. In this paper, we consider these questions, and consider initial answers set out below, in the light of the experience and insights gained so far in FEAST/1. This work has resulted in the development of models of the learning process, presented below using System Dynamics [coy96] formalisms.

It is also interesting to speculate that the learning process, and in particular the rate at which knowledge and understanding can be acquired and applied in changing a software product, might be a limiting factor to system growth – Lehman's Vth Law would suggest it might be.

2. THE QUESTIONS CONSIDERED

We present here initial answers to each of the questions.

2.1. What does 'Learning' Mean in this Context?

An initial definition of this term, used in this paper, is:

> *The accumulation, within the organisation responsible for changes to a software product, of knowledge and understanding relevant to that task. This knowledge and understanding may relate to the product itself, the application domains within which the product is or might be used, and/or how the product is or might be used within that domain.*

2.2. What is being Learned?

What is being learned is the knowledge and understanding described immediately above.

At any point, this knowledge includes a set of assumptions, concerning:

- the product itself,
- the ways in which the product will be used,
- the environments within which the product will be used,
- the purposes for which it will be used,

and so on

As the development organisation's knowledge and understanding grows, what is gained in particular is understanding as to how well these assumptions concerning its nature, use, etc., which are embedded in the product, are standing up as the use of the product changes the world, and as a changing world raises demands for changes to the product. A good example of corporate knowledge about a product which we are investigating as part of the FEAST/1 research is that of the deep and widely held knowledge about VME's life within people currently and formerly in ICL, which (we contend) has helped to maintain the structure and coherence of VME for over 20 years, modification to support many hardware platforms and transition from a proprietary operating system to an open one.

In addition to mechanisms for capturing and storing this vital knowledge, a healthy corporate culture is necessary to maintain that level of interest, involvement, commitment, etc.; do e.g. the process programming people look at corporate inputs at that level?

2.3. Why is Learning Considered to be Inevitable?

Learning during the long-term software process may *inevitably* occur because the use of the product inevitably changes the world, and the world itself changes, and/or because any model of 'the world including the product', implicit in the set of assumptions noted above, is predictive (and therefore likely to contain errors), and also incomplete – as all models are, so at any time, there are both

- some amongst the existing knowledge which is incorrect and has to be unlearned, and
- some new (or previously unguessed) knowledge to be learned.

2.4. How is the Learning Spread Amongst Those who Need to Know It?

- Who needs to know? *people* (specifically the software developers) make changes to a software product. Therefore, the relevant quantity of information held is that which is in the heads of the people making changes to the software.
- It may also be contended that there are other sources of information, which cannot be used directly to change the software product. This information must, unless it is already in the heads of the developers, be added to what the developers themselves know before it can be deployed for system changes. Examples of such information are:
  - product documentation (including source code), and
  - information held in the heads of people not directly involved in modifying the software product, such as sales and marketing people, and help desk staff

2.5. How is that Learning Applied?

What has been learned is applied to *change the software product*. Over time, changed assumptions are embedded in successive versions of any software product which is continuing to be used (Law I) – and these assumptions may in turn be invalidated over time, and so the cycle continues.

What has been learned can also be used as a reservoir of knowledge throughout the development organisation:

- knowledge about similar products can be transferred towards the current product, but this may involve some measure of inefficiency (product differences, transfer inefficiencies etc.); also better staff may be able to transfer their related product knowledge more effectively.
- knowledge in the organisation about the product but not directly related to product changes can also be used; again, there are potential inefficiencies in transferring that knowledge due to the process employed, and further potential inefficiencies due to the quality of the staff used

Other knowledge, in addition to product knowledge, is required to effect changes in a software product. This additional knowledge includes discipline and organisational cultural and technical knowledge [wer96]. It is only product knowledge which concerns us in this paper.

2.6. What Effect Does the Level of What is Learned Have on the Product and its Evolution?

It may be conjectured that, if the current level of knowledge and understanding held within the development organisation is 'too low' (without defining what this means), then evolving the product becomes more difficult; at some stage, the product becomes unmaintainable even if people know about the tools used and some of the details; if the assumption base is lost, then (change specification?) errors will arise if changes are made

A question arise here – to what extent can good documentation resolve this potential problem? An answer may be that, however good the documentation, it may not capture all of the assumptions underlying such aspects of the process and products as specific design decisions.

3. WHERE LEARNING FITS INTO THE PROCESS – A FORMULAIC VIEW

3.1. A Formula for the Product Attributes

One high-level view of the software development process is:

Product = Process (Quality , Resource , Time)

i.e. that the attributes (not further defined here) of a software product of a software process depends on the desired level of quality, the resources applied and the elapsed time allowed for completion of the task.

An equivalent view of the evolution of a software product over multiple releases is:

$Product_{i+1} = Process\ (Product_i\ , Quality, Resource, Time)$

in which the attributes of the i+1th release depends *inter alia* on the attributes of the ith release.

Our view of the software process is that such models must take into account the knowledge *concerning the software product* available and deployed in the development organisation at the time at which changes are made:

$Product_{i+1} = Process\ (Product_i\ , Knowledge_i\ , Quality\ , Resource\ , Time)$

We suggest that any model of the software process which does not take account of the knowledge required in the development staff to make required changes to a software product, and therefore of the knowledge available at that time, any shortfall in the available knowledge and the effects on the product of that process, is to that extent incomplete.

3.2. Defining 'Knowledge'

How might we define the value for 'knowledge' in the above formulae?

We suggest that the knowledge may be considered to be some function of the a number of attributes, including the following, of the product and process, each weighted appropriately:

- the total amount of time spent by the front-line development staff on the product without breaks for any significant length of time to work on other products;[1]
- the total amount of time spent by the front-line development staff on products similar to the product under consideration;
- the knowledge held by people in the development organisation other than the front-line development staff;
- the quantity and quality of development documentation for the product;
- the quantity and quality of documentation for the product other than development documents (sales literature, etc., which may embody assumptions and information concerning the product in non-technical form);
- the knowledge and understanding gained by development organisation staff from help desk interactions with product users; and
- the knowledge gained from other interactions with and publications by users of the product.

A thought arising from this is that it may be useful for development organisations to devise explicit mechanisms for reconciling the mental models of the product held by those inside and those outside the development organisations, especially with regard to the assumptions underlying their perceptions of the product. Development organisations may already doing this informally via user feedback, but is it formalised by e.g. secondment – note the place of product user publications and 'hints and tips' publications; developer organisations may (should?) examine these for ways to improve products.

4. LEARNING IN THE CONTEXT OF SOFTWARE DEVELOPMENT THEORY AND PRACTICE

4.1. Lehman's Laws

There is a close relationship between this paper and Law V (conservation of familiarity); the 'knowledge and understanding' which we are dealing with here is related to the knowledge with which familiarity must, it is claimed in the Law, be maintained. Certainly, the idea that 'product evolution is or can be constrained by lack of knowledge' is a common thread to both.

Law V is currently stated as follows: 'As an *E*-type system evolves all associated with it, developers, sales personnel, users, for example, must maintain mastery of its content and behaviour [leh80a] to achieve satisfactory evolution. Excessive growth diminishes that mastery. Hence the average incremental growth remains invariant as the system evolves.' [Metrics '97] In stating this, the Vth Law effectively also states that some minimum level of knowledge and understanding of a software product, its use and environment is required to support change in that product required by evolutionary pressures. In addition, it has been hypothesised that any changes made without that requisite knowledge and understanding may tend to cause major problems in the product (OS/360 fission – reference). By suggesting that there is a minimum level of knowledge and understanding required, and noting an example of the effects of failure to gain that knowledge, the Law implies that there is some *maximum* rate for knowledge acquisition, limited by aspects of the software development organisation, its structure and personnel.

Taking this requirement for a minimum level of knowledge and understanding to support product evolution down to the level of an individual change to a software product, we can postulate the existence of some minimum level of knowledge required to make that change, and of some level of knowledge and understanding actually directly available to the developers responsible as they perform each action towards the completion of that change. Failures of changes to a software product may thus be due in part or whole to a gap between the knowledge and understanding required to make that change, and the knowledge actually available to those making that change.

In order to understand the evolution of a software product, it therefore becomes essential to track, even if only via crude indicators, the level of knowledge and understanding available to that product's developers as they perform tasks towards the product's evolution. This may allow us to identify points at which lack of knowledge and understanding may have contributed to failures (OS/360 fission), and those in which, despite a change in the product outside the previous trends, sufficient knowledge and understanding have been gained to allow the change to proceed safely (Logica, in which the knowledge and understanding is gained by having one user – the one who paid for the change – use the change live before it is incorporated in the mainstream product).

[1] It may be necessary to include some factor related to the **number** of people on the project as they interact and deepen the sum of their understanding by talking about issues and sharing information. Such a consideration may also become crucial if some of the staff leave, taking the knowledge with them

4.2. Learning-oriented Software Processes

It is also necessary to consider the effect on learning of processes oriented towards maximising the effects of such learning. An example of such a process is one which includes prototyping. However, seen in the context of a software product evolving over time, it can be observed that the prototyping of a software product, especially one in which the functionality is not completely available, in other than its final context of use, is a pretence of real use, not real use, and that for this reason the knowledge and understanding gained may relate to that pretended use of the product, not the effect on the real world of the product's use.

Also consider incremental development mechanisms, which allows the real world use of a subset of the 'final' version of a software product.

5. A QUALITATIVE MODEL OF THE LEARNING CYCLE

5.1. Outline of the Model

As a first step towards a learning-based model of the software process, we describe here a model of the structure of the learning process form which the knowledge described above derives

If we wish to model the way in which *knowledge and understanding about the product* are gained, lost, maintained and deployed in a software product development organisation, the following questions arise:

- how does this knowledge arise?
- how is it stored in the organisation? generally in both people *and* things, so the reservoirs of knowledge in both of these need to be modelled
- how is this knowledge gained and updated?
  - experience gained during the act of changing the product (front-line developers)
  - interacting with customers, potential customers and competitors (other development organisation people such as help desk and sales staff)

  note that
  - we are ignoring the fact that these roles can be held by the same person at times (or other overlaps – e.g. development to help desk rotation); our model is intended to be general to organisations from one person up!
  - new knowledge can include information that some existing knowledge is now obsolete, and should be removed
  - we are talking about (and possibly in the future measuring) the actual knowledge about the product, not what the developer organisation believe they know (incorrect knowledge as well as gaps); there can be a gap between actual and perceived knowledge, and this can lead to project failure (a partial explanation for the London Ambulance Service failure[LAS93] ?)
  - since the use of the product and the environment in which it is used are constantly changing, the knowledge held by the developer organisation concerning the product's use and environment are in a constant state of deterioration, unless they are kept up to date. This is, in a way, analogous to the deterioration of the product's structure over time unless work is done to maintain it (Law VII – declining quality)

- How is this knowledge stored?
  - in the heads of people in the development organisation (note the distinction between those actively involved in changing the software product, and those indirectly involved by reason of their knowing about the product, perhaps only the exterior rather than internal structure and technical information; also note overlap for e.g. pre/post sales consultants, who may know a lot about the externals of a product and enough about the internals to be able to specify changes to it in technical terms, and for people who move around the organisation)
  - in the heads of people outside the development organisation (older generations of VME operations managers are a good example here); this may need to be brought within the development organisation via user groups, help desk interactions, etc.; A development organisation can also get information from complete outsiders, e.g. competitors and external critics of the product
  - in documents (of all types, including source code as well as sales documents, correspondence with potential and actual customers)
- How is this knowledge deployed in changing the product? Such deployment can only occur when decisions are made as to the changes to be made to the product. These decisions are in the final analysis made in the heads of developers. One can conclude that the only directly applicable knowledge and understanding

contributing to these decisions is in the decision makers' heads, and that all other types of storage are off-line 'backing stores', each of which has its own varying

  - delay times
  - resource costs to store
  - resource costs to retrieve
  - efficiency of storage
  - efficiency of retrieval

  Note that some cases retrieval may require more than one step, e.g. implicit product knowledge of outsiders to the development organisation, whose knowledge and understanding needs to be captured (stage 1) and stored (stage 2) within the development organisation

- How might this data be lost other than by some system within the developer organisation deliberately making it obsolete?
  - people forgetting what they know about the product – one might postulate that development staff will forget over time about a product unless they are continuously involved with the product; involvement with other dissimilar products may also contribute to the rate at which product knowledge is forgotten
  - failure to keep the knowledge up to date, such as the retention of obsolete (previously valid, now invalid) information or the failure to observe new use patterns of the product (example: CP/M 1.4 to 2.0 [or other update?] change to delete 'bug' which allowed a reference to a closed file to be reused, bug cleared but database management system developers were using that loophole to ensure that file buffers were flushed whilst keeping a file open – result was old functionality needed to be reinstated at a subsequent release)
  - corruption during the storing and retrieval processes
  - decay over time due to either obsolescence or deterioration of stored media (memory vs. paper vs. magnetic – remember problems of obsolete magnetic media or other!

5.2. Assumptions Made at This Stage of Modelling:

The following assumptions have been made in the analysis leading to the models presented below:

- all knowledge gained is correct; there is no misleading or incorrect information in the information held by the organisation, and there are no contradictions in that information; in real life, some form of information audit function may be required to maintain the correctness and consistency of product knowledge;
- the product knowledge and understanding stored in the development organisation is relevant to the changes being made to the product over time; the question of whether enough knowledge and understanding is available to make a change can be considered on a global basis at this stage (a view more specific to each change may be required later);
- the cycle modelled is an on-going cycle; the feedback structures are due to a perception that the product will continue to be used;
- the product will, if it continues to be used, continue to need changes (Law I); these changes will in turn increase the users' use of the product, often in ways unexpected by the developers, and this will in turn result in demand from the users for further changes to the product;
- the variables for product, knowledge, quality, resource and time are independent sufficiently of each other to allow our calculations of combined metrics to be made; and
- other noise variables in the calculations are sufficiently small to be ignored at this level of abstraction.

5.3. The Top-level Cycle

We present here a top-level influence diagram, showing how the different parts of the overall cycle fit together and how knowledge is gained and lost to the development organisation. Note that

- this is a *feedback* system – information from one release is fed back via various routes into future releases
- the presence (perhaps 'institutionalisation'?) of the knowledge in the organisation may produce a form of inertia, slowing product change
- there are delays (of varying times) at each arrow; all actions take time in this model

5.4. A More Detailed Model

A more detailed model of the cycles which learning causes as knowledge and understanding are extended as part of the software process is set out in Figure 2. As noted for the previous diagram, delays which can be assumed to occur at most of the arrows are not explicitly represented in the figure.

5.5. Comments on the Model

- note the feedback system which is created by modelling the links between the sources and use of knowledge; this supports the view of the software process as being a feedback and learning process expressed in the FEAST hypothesis
- note also the complexity and richness of the feedback system; for example, the variable '`knowledge and understanding embedded in product`' has 44 loops passing through it!

6. SOME OF THE QUESTIONS RECONSIDERED

How has the description of the model advanced our understanding of the software process to enable us to refine the answers to some of the questions raised above?

6.1. What does 'Learning' Mean in this Context?

Learning consists of the making of changes to (not necessarily adding to!) the sum of knowledge available in the developer organisation which is directly related to the product and its use and environment

The models presented here indicates sources from which this knowledge may be obtained, and where this knowledge is stored

6.2. What is being Learned?

knowledge concerning the product and its use and environment, from various sources; this indicates that different sorts of knowledge may be involved, from 'harder' knowledge such as product design documentation and source code, to 'softer' knowledge concerning for instance casual user feedback on annoying niggles with the product in use.

All of the knowledge may need to be deployed to keep the product useful, encourage its use and thus extend its working life

6.3. Why is Learning Considered to be Inevitable?

We have already noted that as the product, its use and environment change, there is more needing to be known by the developer organisation concerning the product, its use and environment. In order to maintain the product's usefulness, change to the product is required. That change must be on the basis of a changed knowledge base in the developer organisation (Law X!). Therefore, if the product is to continue to be used, learning in the developer organisation must occur.

The models show that this learning is a circular process. However a time ordering is in fact imposed on the circular flow, a cycle based on the release schedules of the product. The release date is usually a fixed date, which may be contrasted with the variable dates for adoption of a new version by different users and the time over which user feedback in the form of comments, fault reports, etc. are added to the developer organisation's knowledge of the product.

.. the model shows factors militating in favour of an increase in the amount of knowledge, but there are also negative factors leading to reductions in knowledge of that product

6.4. How is the Knowledge and Understanding Spread Amongst Those who Need to Know it?

Knowledge and understanding are spread amongst those who need to know it by formal and informal systems within the developer organisation. It should however be remembered that the only knowledge which can be applied directly to changing the product is in the heads of front-line developers, note the direct and indirect sources indicated in the model. These need to be captured in some robust way.

Consideration needs to be given in designing any software process to the passing of the knowledge to the place at which it is used and the costs in inefficiency in transferring it to its place of use.

Future work in this area should include specific models of actual industrial software processes with the emphasis on the gaining, storing and use of knowledge.

6.5. How is that Learning Applied?

- applied to realise that changes need to be made to the product, and
- applied in making changes to the product
- these two applications are different, and different mechanisms for storage and retrieval may be appropriate
- all this work points at the need for accessibility for all of the knowledge in the developer organisation

7. QUANTIFYING THE MODEL

7.1. Introductory Notes

- it may be possible to use System Dynamics as a mechanism for quantifying the models presented previously. We discuss here some of the specific details of such a quantification, and issues and problems which might arise. It is important to note that we are not claiming that a route can be achieved to the quantification of all software processes which will allow comparisons to be made for a set of these metrics with (in increasing order of unlikelihood) other products from the same developer organisation, other processes from the same developer organisation, and other organisations. We are only claiming that a route may be found to examine trends in *one product* in *one organisation* over time, which can be compared with other metrics for that process (cost, timescales, test fault rates, absolute sales figures, market share, …) and product (functionality, size, field fault rates, …)
- the single-product approach also allows certain aspects of the process, such as the corporate culture, to be factored out as constants if they are unchanged over the time period being examined, so simplifying this first attempt at quantification

7.2. Why do It?

- it may be of direct use to software development organisations to be able measure the knowledge and understanding available to product developers over time as a metric (possibly calculated and combined from a number of actual measurements) whose trends can be related to the trajectory of other measures of the software product over time. Such a measure may in time become a formalised risk assessment factor in deciding whether to incorporate a particular change in a software product. However, to reach this stage research would also have to be performed into the localisation of product knowledge and understanding relative to the localisation within the product of the change, so that the knowledge and understanding available could be related to the specific knowledge and understanding required for that change to be made safely.
- test validity of Law V (conservation of familiarity), especially if the metric is related to the amount of knowledge *relative to the size of the product*
- as well as how it might be done, some possible difficulties are discussed in this section

7.3. What Needs to be Included in the Model

- The main loop as given before should be the basis of the model
- The efficiency of deployment of the knowledge may also have to be considered. If this factor is deemed to have been constant over time, then it can be ignored as a constant. However, if it is suggested that the efficiency factor has changed, due say to improved technology, better training etc., then it may have to be calculated and the changing value reflected. Sub-factors for the efficiency factor may include:
  - quality of staff
  - motivation of staff
  - quality of process
  - quality of mechanisms for sharing information between people
- changes in knowledge (increases as developers learn about the product by working on it, decreases as unused knowledge in memory decays and knowledge is made obsolete by changing product use and environment). These changes are probably not linear over time; it my be reasonable to use a first-order exponential smoothing, as used in a learning example from Coyle [coy96], and in the software process modelling world by Abdel-Hamid [abd91, p.65]

7.4. What Needs to be Quantified – Theory

- the essence in the model, what we are actually trying to track over time, is *knowledge and understanding*
- to examine this, we need to look to the useful knowledge and understanding contained in a heterogeneous collection of documents and in the heads of people
- thus we are trying to measure the knowledge and understanding:
  - in the heads of people

  - explicit and implicit (assumptions) in documents (development and sales etc.)
  - explicit and implicit (assumptions) in the product (development and sales etc.)
- the measure of learning should be a *ratio* measure, to allow meaningful comparisons to be made between different values over time.
- … note that SD is good at quantifying qualitative aspects to give reasonable results (Abdel-Hamid?)

7.5. What Metrics Might We Use or Need? Some Practical Issues

- what measures *in the real world* can we use as aliases to the theoretical metrics whose behaviours we wish to examine?
- also need *product* metrics, against which to compare the trends we get from the knowledge modelling
- The indirect sources of knowledge mentioned above (non-development staff, documentation) would also have to be calibrated relative to each other, to allow values aliasing knowledge and understanding gained from different sources to be compared and combined.
- The values obtained for the different sources would be subject to divider adjustments (to reflect inefficiencies in storage and retrieval processes, and for staff quality) and possibly delays before they could be deployed. These divider factors would themselves have to be calibrated. Documentation may need to be measured in a multi-dimensional way – size, quality (understandability, etc.); there may be existing measures to be used
- what are meaningful and useful input metrics, in terms of what might actually be measurable, and thus delivered to the simulation model as fixed input data over time? Such metrics might include:
  - *person time* spent on some particular activity, such as developing the product, developing similar products, providing help desk services, selling the product, creating and updating documents, etc.
  - the *quantity* of product technical and/or sales documentation, perhaps modified for any change in quality over time; but there is a difficulty here in translating quantities of documentation into measures to be combined with time spent on the product – what is the conversion factor?
- the easiest way to proceed might be to examine the behaviour of suitably weighted measures of *actual person time*[2] spent on different activities, including:
  - developing the product (with different weights for current uninterrupted work and previous experience with the product),
  - developing similar products,
  - providing help desk services (with some weighting for how the load on the help desk changes over time),[3]
  - selling the product, and providing pre- and post-sales support, again allowing for uninterrupted, prior and similar product experience
  - creating and updating documents, which has to provide direct knowledge to the documenters (cf. developers, same modifiers) *and* organisational knowledge in the form of information which can be accessed in the future if required (quality modifiers?); note that the 'time spent' measure to be used here to capture increased knowledge and understanding is that of authoring and (possibly) editing only, not production and distribution![4]
  - performing market awareness and product comparison exercises within the developer organisation
  - *user time* spent examining, testing, using, experimenting with the product; this, since we are unlikely to have the numbers of people at each user or evaluator, may be some measure of *numbers* of organisations using or evaluating the product; generally, we might assume that more users will tend to find more ways to use (and break!) the product, thus increasing the throughput of those feedback loops – measures of numbers of user/outside evaluator organisations over time might therefore be a valid metric here. A question here might be how the size of the product affects this metric
  - overall, we are using *experience* as an alias for *knowledge and understanding*

[2] This should be measured in consistent, convenient but not necessarily exact units, bearing in mind the lack of precision in the quantification of the model and the timescales over which product and process behaviour are being examined – a suitable measure might perhaps be full-time equivalent staff months.

[3] The process of help desk interaction could be considered to be a learning event for the *user organisation* as well as the development organisation, increasing the level of knowledge and understanding on both sides. The effect of help desk interactions on user organisation knowledge and understanding is currently not included in the model, such events being considered to be folded into the user learning process over time.

[4] The current Vensim model wraps the updating of documentation into the updating of the product; no differentiation is made between time spent writing documentation, and actually designing and implementing product changes. This ensures that increased personal learning in documenting is included in the measure captured, but excludes the possibility of having different weightings for the two activities.

- *modifiers* are needed to scale each of these time measures to each other and allow combined measures of knowledge and understanding to be calculated; these modifiers need to allow for:
  - product size, as discussed above
  - how people learn; tailing off (use an SD *smooth*?) of gains in knowledge and understanding as each person learns more about the product space
  - organisational efficiency in capturing knowledge and understanding, differentiating perhaps between formal and informal channels of communication
  - organisational efficiency of retention of knowledge and understanding (staff turnover, document filing, keeping knowledge and understanding up to date
  - organisational efficiency in retrieving knowledge and understanding when required
  - organisational efficiency in applying retrieved knowledge and understanding to product changes
  - quality of product, documentation and
  - quality of staff, in the form of a modifier to the time taken to acquire knowledge and understanding

  note that
  - if we are examining the behaviour over time of a single product within a single organisation, we can more easily justify the use of relative values for these modifiers, rather than searching for absolute values for modifiers which would need to be able to be compared across products and organisations
  - if the values for these modifiers seem not to change over time for this product and organisation, single values may be employed in calculations for the model rather than changing or calculated factors
  - organisational factors influences the efficiency of capturing, storing, retrieving and applying knowledge and understanding might include: (taken from [mar58], p.163–166)
    - the existence and extent of a common language between staff members in different areas (development, support, sales, etc.)
    - the success or otherwise of the organisation in copping with the intangible nature of software products in defining that common language
    - the efficiency of the formal and informal networks of communications between staff members in different areas of the development organisation, not only in transferring knowledge but in finding out that a particular person or role may be expected to have that knowledge

    these may also apply to each actual and potential product user and evaluator organisation, but, again, since we would lack information concerning details of the internal workings of these organisations, and since there are a larger number of them than development organisations, we are justified in assuming some measure of homogeneity between user and evaluator organisations and thus ignoring (factoring out) differences in calibrating the model
- the outputs of the model, whose behaviour overtime might be compared with other process and product metrics, would include:
  - the trend over time in total of developer organisation knowledge and understanding of the product, its use and environment *available* for inclusion in the product
  - the trend over time in total of developer organisation knowledge and understanding of the product, its use and environment *actually included* in the product
- the values for these model outputs might be compared with trends in other process and product metrics in such ways as:
  - comparing the knowledge and understanding incorporated in the product with the actual rate of product enhancement requests; here, an increase in the trend of the model output might be expected to result in an increase in the enhancement request rate
  - comparing the trend in knowledge and understanding incorporated in the product divided by the trend in change of size (measured in units with semantic meaning, such as module counts) of the product – forming some measure of the average knowledge and understanding in enhancements to the product – with the reporting rate for analysis and design errors (implementation errors may be related to other issues, such as pressures on timescales – see below); it might be postulated that the less knowledge and understanding incorporated in a change to the product, the more likely it is that errors of understanding will be made
- possible weaknesses in this approach may include:
  - the use of time spent on a task as an alias for learning; this may create problems, for instance, were implementation error rate trends to be compared with trends in available or applied knowledge produced as model outputs, since management pressures on timescales and resources might reasonably be postulated to result in increased implementation error rates without considering any shortfall in knowledge and understanding
  - rationalising the ability to create combined measures (particularly when person (internal) time and organisation (user) time measures are combined, and determine what they *mean*)

  - difficulty in calibrating the modifiers used to weight the different time values in the model when combining them; some of these may be available from the literature (are there studies into, for instance, differences made in time spent changing a program depending on the quality of the code, the availability and quality of documentation, etc.)

- note that using absolute measures of time spent in a particular activity does not make sense, since there must be an effect due to the size of the product on which the time is spent; less time on a smaller product may result in the acquisition of more of the potential knowledge on that product than a greater time spent on a larger product
- A example of a meaningful ratio metric for the amount of knowledge and understanding held relative to the size of the product is of the form of man months of experience on this product / product size – some form of 'Knowledge Density' (KD)?
- As a first approximation to the full KD computation, it may be useful to examine the behaviour over time of the following calculation:

  *average time of current team members on project * number of team members / size of product*

  We may be able to do this as part of the FEAST/1 work, using the data collected for that project as a starting point

8. FUTURE WORK

- relate the models presented here to actual software processes in industrial contexts, modifying the models either to a more accurate generic picture if possible, or if not to each real world context
- produce qualitative conclusions, such as routes to capturing all of the information regarding a product, storing it in a retrievable fashion and keeping it up to date
- also extending the model to look at the costs and losses of storing the knowledge in different ways, to help optimise that storage and retrieval
- quantify the model as suggested above, and, in specific individual product processes, calibrate the model and run simulations to examine the explanatory or predictive power of the model in relating product knowledge to other product or process metrics such as cost, quality and functionality
- examine the quantified results and compare then to existing quantitative models of software evolution, in particular Turski's Inverse Square Model [tur96]
- incorporate the work into mainstream FEAST/1 white box work – extend the current model and incorporate it into the existing models of the software process, examine the combined models for feedback loops and particularly feedback controls, critique and try to improve the feedback structure, with possible implications for organisational changes and product/documentation at any level to improve the retention and accessibility of the organisational knowledge
- It may be possible to correlate a quantified track of KD, possibly with a time delay, against other trends in a product's metrics, such as the existing size metrics and fault rates. This may be especially interesting when these metrics diverge from smooth paths.
- it may be possible in the long term to develop a risk metric based on a comparison of the trends in the size of a software product based on the expected size of any proposed change and the trend in underlying knowledge and understanding of the product, its use and environment
- This model may also be related to other ideas on, and models of, the software process, e.g. work on the Learning Model of the software process at the Tavistock Institute
- In the final analysis, the software process can be viewed as being about holding information about the software product, and deploying that knowledge as required.

9. SUMMARY AND CONCLUSIONS

From the viewpoint taken in this paper, the software process involves, *inter* alia, the gaining, storing and using of knowledge and understanding about a software product.

Real world software processes must be aware of, and optimise, this learning process; the novelty of this work is not the realisation of the importance of product knowledge and understanding, but its placing of that realisation in the context of process modelling and, by the future use of System Dynamics techniques which are designed to allow the quantification of qualitative attributes of human-centred processes, the prospect of placing it in the context of *quantitative* process modelling, allowing a brick to be put into the wall of a theory of the software process

This approach will also form an important part of the on-going FEAST research programme, building models of the long-term software process working towards a theory of software product evolution

ACKNOWLEDGEMENTS

The work reported here has been supported under EPSRC grant number GR/K86008.

REFERENCES

[abd91] Abdel-Hamid T and Madnick SE; *Software Project Dynamics*; Prentice-Hall, Englewood Cliffs, NJ; 1991

[ara93] Aranda RR, Fiddaman T and Oliva R; *Quality Microworlds – Modeling the Impact of Quality Initiatives over the Product Life Cycle*; American Programmer; May 1993; pp.52–61

[coy96] Coyle RG; *System Dynamics Modelling – a Practical Approach*; Chapman and Hall; London; 1996

[LAS93] *Report of the Inquiry into the London Ambulance Service*; South West Thames Regional Health Authority; London., 1993

[mad96] Madachy RJ; *System Dynamics Modeling of an Inspection-Base Process*; *in* Proc. ICSE 18; pp.376–386

[mar58] March JG and Simon HA; *Organizations*; Wiley; New York; 1958

[mml94] Lehman MM; *Evaluation, Feedback and Software Technology*; *in* Ghezzi C (ed.); Proc. 9th International Software Process Workshop; IEEE Computer Press; 1994; pp.134–137

[mml96] Lehman MM and Stenning V; *FEAST/1: Case for Support*; Department of Computing, Imperial College London; EPSRC Proposal; March 1996

[smi93] Smith B *et al.*; *The Software Process Model*; Draper Laboratory; Cambridge, Mass.; technical report CDSL-R-2515; June 1993

[tur96] Turski W; *Reference Model for Smooth Growth of Software Systems*; IEEE Trans. on Software Engineering; vol. 22, no. 8, 1996

[wer96] Wernick PD; *A Belief System Model for Software Development*; PhD Thesis; Department of Computer Science, University College London; April 1996