A dynamic simulator of software processes to test process assumptions

Paolo Donzelli, Giuseppe Iazeolla *

Laboratory for Computer Science and CERTIA Research Center, University of Rome at Tor Vergata, 110, Via Di Tor Vergata, 00133 Rome, Italy

Received 21 October 1999; received in revised form 7 December 1999; accepted 9 February 2000

Abstract

Validation testing of software processes may provide both qualitative and quantitative suggestions to understand the ways to change the software process to improve its quality, or the ways to achieve specific organisation’s needs. In many cases, however, such understanding has to be done without affecting the actual environment. To this purpose, this paper introduces the use of process simulators for validation testing. To be effective, however, simulators must combine the ability to sustain the complexity of the modelling problem with the so-called dynamic estimation capability, that is the capability of representing the dynamics of the simulated process. To achieve such objectives, the introduced simulator is based on the association of three conventional modelling methods (analytical, continuous and discrete-event) into a unique hybrid multi-level new model, called dynamic capability model (DCM). The paper applies DCM in the context of a waterfall-based software process to study the effects of three different quality assurance management policies on given process quality attributes, as effort, delivery time, productivity, rework percentage, and product quality. The verification of the simulator representativeness is also performed by reproducing empirically known facts in the process behaviour.

Keywords: Software process modeling; Simulation modeling; Hybrid simulation

1. Introduction

Validation testing is the activity through which the capability of a given process to satisfy user needs is tested.

Validation of process assumptions is one form of validation testing. It consists of ensuring that the effects of introducing variants to the current paradigm of software development are the desired ones. For example, a project manager may want to be sure, before adopting an extra review activity, that this will not lead to any risk of overrun; a testing manager may want to verify to have enough staff before committing himself towards higher test performances; a process modeller may want to estimate the amount of process overhead that could be generated by introducing higher concurrency levels among process activities.

Software companies face the problem of validating process assumptions whenever they are dissatisfied with the current productivity or quality levels and are willing to experiment new development policies.

Different assumptions on many parameters of the software process can strongly affect its final quality. Such parameters range from the structure adopted for the particular process according to the chosen paradigm (number and type of phases and activities, nature of the exchanged artifacts, etc.), to the allocation of defect detection resources (personnel, tools, and methods) along the life cycle, to the priority policies assumed for the different tasks (defect correction, requirements change, etc.).

The use of process models is an effective means to test and validate new assumptions on parameter choices. To be representative of the real system, however, the model must be capable of reconstructing process trajectories over time under different what–if conditions, and of estimating process’s outcomes in a perturbed environment. A feature this paper calls the dynamic estimation capability.

To better understand such a feature, one may consider that models conventionally used by the software community to analyse and predict process behaviour are of analytical average-type and do not generally hold
such capability. Examples of such models are the function point-based models (Albrecht, 1979) to estimate the final product size and associated effort; the COCOMO model (Boehm, 1981) to estimate schedule and effort for a given software project; the Rayleigh model (Putnam and Meyer, 1992) to shape the staffing profile; the reliability growth model (Fenton and Pfeifer, 1997) to estimate the testing effort required to achieve a desired quality. Besides lacking dynamic estimation, and providing only "average" estimates on a strict subset of the quality attributes, these models separate the effects of such attributes (e.g., schedule from defect density), and do not permit the analysis of the process behaviour in a perturbed environment (e.g., changes in initial requirements, staff reductions, etc.).

Giving a process model the dynamic capability generally requires the introduction of so complex relationships between internal and external variables, that simulation approaches have to be adopted, since mathematical ones require the introduction of many simplifying assumptions, and thus the resulting model is scarcely representative of the real system. However, existing process-simulation techniques also suffer by drawbacks, in that they only enhance some process aspects to detriment of the others. This is because such techniques are either of discrete-type, discrete-event queuing based (Hansen, 1996), or of continuous-type, system dynamics based (Abdel-Hamid and Madnick, 1991; Calavaro et al., 1995; Rus et al., 1998), and only rarely give a combination thereof.

It is of view of this that to give a model the dynamic capability property (i.e., the real system representativeness) one has to combine all the three above-mentioned modelling methods (the average-analytical modelling method, the discrete-type and the continuous-type one) into a hybrid modelling method. In this view, the paper introduces a predictive hybrid model, called the dynamic capability model (DCM). The hybrid method is applied in DCM according to a two-level abstraction framework. At the higher abstraction level, the discrete-event method is used, while the analytical and the continuous-type methods are used at the lower abstraction level. This also motivates the reasons why each approach is used at each level. Indeed, the software process shows both discrete system aspects (start/end of an activity, reception/release of an artifact by an activity) and continuous system ones (resources consumption by an activity, percentage of developed product), and thus the proposed modelling approach provides a way to hierarchically take into account such different aspects.

The paper contribution is centred on the model production methodology. In other words, the paper gives a method to produce a model that is highly representative of the real system. In any modelling study, the subsequent step of model production is model evaluation. As stated above, in this paper model evaluation is performed by use of simulation. To this scope, the QNAP2 package is used (Simulog, 1986), although any other simulation package could have been adopted. We have chosen QNAP2 since it provides convenient language primitives to support the description of the hybrid model.

It is generally argued that simulation solutions are unlikely to be able to give exact forecast of the real process behaviour. However, it is to be view of this paper that they nevertheless give projections on how the process would behave under given assumptions on external and internal factors. They stimulate debate and provide a way to learn about how to improve process quality. To sustain this view, some application examples are presented, which predict and analyse the effects of process-management factors (reviews and testing effectiveness) on process quality attributes as effort, delivery time, productivity, rework percentage and product quality.

This work is one of the results of the joint co-operation between the University of Rome “Tor Vergata”, the Enterprise-University Consortium CERTIA, the Software Engineering Laboratory of the University of Maryland and the CERC Research Center of the University of West Virginia, on the “Concurrent Engineering and Simulation Modelling in Software Process Optimisation” enterprise-university project.

The paper is organised as follows. Section 2 gives a brief overview of DCM. Section 3 describes an example use of DCM to study process quality. Section 4 deals with the verification of the simulator validity, and Section 5 gives conclusions and plans for future work.

2. DCM for the waterfall paradigm

Software processes based on the waterfall paradigm are taken into consideration in this paper, and modelled in DCM. According to such paradigm, the software process (illustrated in Fig. 1) consists of a series of sequential phases, and the software product is the conclusive artifact of a series of intermediate artifacts, named requirements, specification (SP), high-level design (HLD), low-level design (LLD), code, system-tested code and acceptance-tested code. Such artifacts are also referred to as primary artifacts.

Although phases are sequential, their respective activities can run concurrently, because of the simultaneous execution of work activities (that generate primary artifacts) and rework activities (necessary to fix defects or to introduce requirement modifications). Artifacts generated by the rework activities are referred to as secondary artifacts. They are instead named defect reports or correction reports if generated by activities aiming at fixing defects. They are instead named changes or...
increments if generated by activities that introduce modifications due to requirements instability. The waterfall process thus consists partly of sequential and partly of concurrent activities.

Activities are distinguished into development activities, and testing activities. In Fig. 1, the development activities are the SP, the HLD, the LLD, and the implementation (IMP) activity. The testing activities are the system test (ST), and the acceptance test (AT). The (primary or secondary type) artifact’s various activities yield are reported on the arrowhead sides in Fig. 1.

Fig. 1 process is translated in DCM according to a two-level abstraction framework: the higher and the lower abstraction level, described in Sections 2.1 and 2.2, respectively.

The process quality attributes taken into account by DCM are effort ($W$), delivery time ($T$), productivity ($P$), rework percentage (RWK), product defect density (DFD) and many sub-attributes thereof (process staffing profile, staffing profile over single activities, duration of each phase, final product size, etc.). However, for the sake of conciseness, in this paper, we concentrate on the study of only a few of them.

2.1. The DCM higher abstraction level

At the higher abstraction level, the discrete-event modelling method is used. The process is modelled by a discrete-event queueing network. The queueing model is a direct replica of the software process. Service stations are used to represent activities and sub-activities, whilst circulating customers are used to represent artifacts that move from one activity to another and that are enqueued on the entrance of a given activity and wait for services.

Fig. 2 illustrates the queueing network used to model the HLD activity. The main service stations are the “work station”, the “external rework station”, the “internal rework station” and the “review station”.

The work station simulates the development of the HLD artifact on the basis of the demand submitted in input by the SP artifact.
Based on the demand in input for SP changes, or SP increments, the external rework station simulates the modification of the already released HLD artifact, and yields the corresponding output artifacts (HLD changes and HLD increments). Similarly, based on the demand in input for SP correction reports or HLD defect reports, the internal rework station simulates the correction of the released HLD artifact, and yields the corresponding HLD correction reports.

Finally, the review station simulates the review performed on the HLD, the HLD changes, and the HLD increments artifacts. No review is performed on HLD correction reports, assumed with no defects. In other words, it is assumed that the correction activities (simulated by the internal rework station) inject no defects.

The “start”, “release” and “store” stations in Fig. 2 are assumed to be zero service-time stations, since they perform just co-ordination activities. In particular: the “start station” directs the input artifact to the appropriate service station, depending on its type, whereas the “release station” and the “store station” take care of the releasing of the artifacts.

The HLD, the HLD changes and the HLD increments are released by the release station only if no defects have been found by the review station. If, however, some defects have been found, the release station creates the corresponding defect reports (e.g., HLD and SP defects reports) and sends them back to the activities responsible for the defects. The faulty artifacts are then sent to the store station, where they are held until all the correction reports corresponding to the released defect reports are received.

2.2. The DCM lower abstraction level

The lower abstraction level gives the implementation details of the service stations (or sub-activities) introduced at the higher abstraction level. The analytical and the continuous modelling methods are used at this level. In particular, each sub-activity is modelled either by an analytical average-type function, or by a continuous type time-varying function (or by a combination thereof). Such functions are used to express the amount of resources (e.g., personnel), or time, or effort (person-week) that service stations use to simulate the corresponding sub-activities.

Fig. 3 shows the implementation details of the work station, one of the main service stations depicted in Fig. 2, and of its corresponding input and output artifacts. The station simulates the development of the HLD artifact, starting from the SP artifact.

The SP and HLD artifacts are described by a set of four attributes: name, size, development effort and defectiveness. Attribute’s name and size are of immediate evidence. The attribute defectiveness is described by an array whose /th element is the amount of defects injected into the artifact by the /th development activity (j = SP, HLD, LLD, IMP). The attribute total development effort (W1 for the SP and W1 + W for the HLD) is the total effort that has been spent to develop the artifact itself since the beginning of the process. Thus, it encompasses also the effort spent to develop all the artifacts from which it has been derived.

The values of the attributes of the HLD artifact, together with the amount of time, T, (work station service time), and of personnel over time, E(t), required to develop such an artifact, are derived as illustrated in Fig. 3. All these quantities may have random deviations, and are therefore simulated according to Gaussian-like probability distributions. More in detail, the average size of the HLD artifact is first derived from the size of the SP artifact by use of the COCOMO-like size estimator block. The corresponding random size is then obtained by use of the COCOMO-like time estimator block to obtain the random development effort (W). That is, as shown by the shaded area in Fig. 3, the effort simulated by the

![Fig. 2. Higher abstraction level of the HLD activity.](image-url)
work station to develop the HLD starting from the SP.

The random development effort \( (W) \) is then added to the value of the total development effort attribute of the SP artifact \( (W1) \) to obtain the value of the corresponding attribute of HLD \( (W1 + W) \).

On the basis of \( T \) and \( W \), the amount of required personnel, \( E(t) \), is finally obtained using the Rayleigh function (Putnam and Meyer, 1992). According to Putnam’s assumption [3], the HLD artifact is released when \( E(t) \) reaches its peak. Moreover, unlimited staff availability is assumed. In other words, it is assumed that the staff pool in Fig. 3 can always supply the personnel necessary to fit the \( E(t) \) curve demand for personnel. DCM, however, can easily accept more realistic assumptions on finite staff pools.

The amount of defects injected into the HLD artifact (injected defects, ID) is obtained from the injected defect estimator block, as a by-multiplication of the random size of the HLD and the expected defect density (defects per unit of size). Defect density is a parameter used in DCM to summarise the effects of various factors (personnel skill, team structure, supporting tools, programming language, product type, etc.) on the defectiveness of a given development activity. DCM, however, can easily accept more elaborate defect injection models, as for example models in Stutzke et al. (1998).

The derived ID is then summed to \( D1 \) (SP defectiveness) to obtain the HLD defectiveness.

More details on the analytical derivations of the functions used to model this station (and all the other stations in Fig. 2) are in Donzelli and Iazeolla (1996), Donzelli (1997) and Donzelli and Iazeolla (1997).

3. Example use of DCM to study process quality

The DCM simulator can be used to predict and analyse the effects that various assumptions on process
parameter choices can have on the process quality (e.g., effort, delivery time, and productivity). Even if, in many cases, such effects are known facts at qualitative level, they are made quantitative by use of the DCM simulator. In this section, the quantitative study of the so-called “find as much as early as possible” defect detection strategy is made. In qualitative terms, it is already known that such strategy improves the process schedule, effort and the final product quality. The use of DCM will provide a quantitative evaluation of such advantages.

In such a perspective, DCM will be used to study the effects of three different defect detection policies (P1, P2 and P3) in a software development scenario with stable requirements. Stable requirements meaning that the size of the initial requirements (assumed to be of 1500 function points) does not change during product development.

The three policies are characterised by different allocations of the defect detection resources along the life cycle, however yielding the same final product quality (simply measured in DFD). In DCM this can be expressed by assuming different defect detection effectiveness (DDE) for the process defect detection activities (SP-review, HLD-review, LLD-review, IMP-review, ST and AT). In fact, DDE is a DCM input variable that can be used to specify the detection effectiveness of a review or testing sub-activity in terms of percentage of removed defects.

The values of DDE adopted for P1, P2 and P3 are reported in Table 1.

In summary, it is assumed that in P1 (or early detection policy) the DDEs are higher in the initial activities of the life cycle, in P2 (or middle detection policy) the DDEs are higher in the middle activities of the life cycle, in P3 (or late detection policy), the DDEs are higher in the final activities of the life cycle.

Comparison is made by use of the DCM simulator, analysing how the values of the attributes $W$, $T$, $P$, and RWK (DFD is constant) change from P1 to P2 to P3.

Figs. 4 and 5 illustrate the simulation results for the personnel over time $E(t)$ in the early and late detection policies. They show that when the early detection policy is applied a reduction of effort $W$, represented in Figs. 4 and 5 by the shaded area, and of delivery time $T$ is obtained. In particular, the effort moves from 581 to 519 person-week, whereas the delivery time moves from 102 to 82 weeks.

Furthermore, it can be observed that in the early detection policy case, more time and effort are spent during the development phases (SP, HLD, LLD, and IMP phases) rather than during the testing ones (ST and AT phases). On the contrary, when the late detection policy is applied, more time and effort are spent during

<table>
<thead>
<tr>
<th>Table 1</th>
<th>Defect detection effectiveness for early, middle and late policies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Policy</td>
<td>SP-review (%)</td>
</tr>
<tr>
<td>Early detection</td>
<td>95</td>
</tr>
<tr>
<td>Middle detection</td>
<td>10</td>
</tr>
<tr>
<td>Late detection</td>
<td>5</td>
</tr>
</tbody>
</table>

Fig. 4. Personnel over time for the early detection policy.
the testing phases rather than during the development ones.

A better understanding of the process behaviours under the different defect detection policies required us to carry out further experiments. To this purpose, Fig. 6 has been first obtained by use of DCM. This picture shows the defect detection patterns for the defects injected by the HLD activity (for the early and the late detection policies). The histogram bars along one column (i.e., late or early policy) give the number of defects that have been injected by the HLD activity and have been detected by the subsequent defect detection activities (HLD-review, LLD-review, ST and AT). For example, the IMP-review bar in the “late” column indicates that about 50 defects, injected by the HLD activity, have been found during the IMP-review in the late detection policy case. Thus, the heights of the histogram bars along one column sum up to the total amount of defects which have been injected during the HLD activity and which have been detected by the subsequent defect removal activities (e.g., 340 in the Fig. 6 case).

In summary, Fig. 6 shows that in the early detection policy most of the defects are discovered and reworked...
locally, i.e., during the same activity that injected them, whereas, in the late detection policy, most of the corrections are not performed locally, but during the testing phases and this contributes to move time and effort towards the testing phases (as in Figs. 4 and 5).

In addition, although the total number of removed defects is the same (340 in Fig. 6), in the late detection policy case they are discovered later, leading to a higher consumption of resources during the defect correction cycles. In terms of process quality, this contributes to the higher effort, delivery time, rework percentage (from 17% to 25%) on one side, and to the lower productivity (of the 13%), on the other, for the late detection policy case in comparison with the early one. A further validation of the observed differences among the three policies can be found by considering the total amount of effort spent in removing defects. In particular, to this purpose, further simulation experiments have been carried out, which show that, moving from the early to the late detection policy, the total effort spent by review and testing activities increases by about 18%.

To further illustrate the utility of DCM, Fig. 7 has been obtained, which gives a synthetic view of the normalised values of the process quality attributes for all the three policies (P1, P2, and P3). It can be seen that moving from the early to the late detection policy, the effort ($W$), the delivery time ($T$) and the rework percentage ($RWK$) increase, whereas the productivity ($P$) decreases. As assumed, the final product quality (DFD) at the life cycle end is the same for the three policies.

4. Verification of simulator validity

Verification of simulator validity is obtaining a good level of confidence in the representativeness of the simulation model against real-life situations. To verify the simulator, experiments are to be carried out to reproduce empirically known facts in the simulation experiment. Many experiments have been carried out to verify the model. The one this paper presents is to reproduce the effects of the instability of the requirements on the process performance. It is empirically known that requirements instability degrades the process delivery time ($T$), the effort ($W$), the productivity ($P$), the rework percentage ($RWK$) and the final product quality (DFD). If the simulation model reproduces such results, we get better confidence in its representativeness of real-life situations.

The verification experiment consists in comparing two possible software development scenarios. In the first scenario, a stable set of requirements is assumed. That is, it is assumed that the initial requirements do not change during the product development. In the second scenario, a certain amount of instability in the requirements is simulated, by allowing the user to add new requirements, or change some of the old ones. In particular, in the first scenario, a requirements artifact of 1500 function points is fed in input to the process model. In the second scenario, increments and changes artifacts are regularly fed into the process, to simulate a continuous requirements increment and change activity. Over the development time, the requirements grow from the initial amount of 1500FP to an amount of 1500FP + 20%, while the 15% of the initial requirements are changed.

Fig. 8 illustrates the effects of such instability over the staffing profile. The profiles are similar in the two sce-
narios at the beginning of the project and diverge when the effects of the new and changed requirements increase. This confirms the empirical expectation that when requirements are unstable a general increment of staff over time is produced.

This immediately affects the main process attributes as summarised in Fig. 9, where they are compared with the corresponding attributes obtained in case of stable requirements.

The simulator results confirm the empirical expectation that due to requirements instability, a substantial increase of effort \((W)\) and delivery time \((T)\) are introduced, and provide a quantitative prediction of such an increment (38% and 60%, respectively). Fig. 9 also confirms the expected decrease of the process productivity \((P)\), the increase of the rework percentage \((RWK)\), and the deterioration of the final product quality \((DFD)\). Indeed, the rework percentage has more than doubled (a 150% increase), the productivity has clearly dropped (a 21% decrease), and the defect density of the final product has increased (a 66% increase).

5. Conclusions

Understanding and managing the quality of the software process is the goal of many organisations. On the other hand, the modelling process is quite complex. In order to deal with such a complexity, a combination of three methods (analytical, continuous and discrete-event), into a unique hybrid multi-level modelling methodology with dynamic capabilities, is proposed.

The methodology is applied to a waterfall-based software process to produce a DCM, which is then used to predict the effect of process-management factors (e.g., reviews and testing effectiveness) on some process quality attributes, as effort, delivery time, productivity, and product quality. Applications of the model show that simulation results can provide both qualitative and quantitative suggestions about how to change the software process to improve its quality or to fulfill the organisation’s goals.

Plan for future work includes the extension of the model to less conventional process paradigms, such as the spiral paradigm and the concurrent engineering paradigm.

Acknowledgements

This work was partially supported by the CERTIA Research Center, and by the UNIROMA2-UMD-WVU Co-operation agreement on Concurrent Engineering and Simulation Modelling in Software Process Optimisation.

References


**Paolo Donzelli** is an Advisor with the Department of Informatics, Telecommunications and Statistics of the Office of the Prime Minister, Italy. Formerly, he served as an engineering officer with the Italian Air Force, and then he was with Cranfield University (UK), as Senior Research fellow. Dr. Donzelli received a doctoral Laurea degree from the University of Naples “Federico II” (Italy), a M.Sc. degree from the Cranfield University (UK), and a Ph.D. degree from the University of Rome “TorVergata” (Italy). His research interests include software process improvement, requirements engineering and business process modeling.

**Giuseppe Iazeolla** is full professor of Computer Science, Software Engineering Chair, Faculty of Engineering, University of Rome Tor Vergata (Italy). His research is in the areas of software engineering and information system engineering, in relation to system performance and dependability modelling and validation.