Fraunhofer Institut
Experimentelles
Software Engineering

Using Simulation to Visualise and Analyse Product-Process Dependencies in Software Development Projects

Authors:
Dietmar Pfahl
Andreas Birk

Accepted for publication in
Proceedings of PROFES 2000,
June 20-22, Oulu, Finland

IESE-Report No. 013.00/E
Version 1.0
February, 2000

A publication by Fraunhofer IESE

Fraunhofer IESE is an institute of the Fraunhofer Gesellschaft.
The institute transfers innovative software development techniques, methods and tools into industrial practice, assists companies in building software competencies customized to their needs, and helps them to establish a competetive market position.

Fraunhofer IESE is directed by
Prof. Dr. Dieter Rombach
Sauerwiesen 6
D-67661 Kaiserslautern

Abstract

The core element of the PROFES improvement methodology is the concept of product-process dependency (PPD) models. The purpose of PPD models is to help focus process improvement activities to those development technologies and processes that are most effective with regards to achieving specific customer-defined product quality goals. This paper describes how system dynamics simulation models can be used to check the plausibility of achieving positive effects on software product quality when implementing improvement actions derived from PPD models. Basically, this is done through extending an existing generic software project simulation model with structures that represent expected local cause-effect mechanisms of the PPD models. By running simulations with the extended software project simulation model, the potential effects of the PPD models on product quality can be investigated at low cost before conducting pilot applications in real projects.

Table of Contents

1 Introduction

Software industry is constantly facing increasing demands for quality, productivity, and time-to-market. At the same time, increasing complexity of software products and projects makes it ever more difficult for software developers and managers to improve performance. One reaction to this challenge has been the – now widely accepted – practice of initiating and conducting continuous software process improvement (SPI) programmes.

Triggered by the seminal work of Watts Humphrey [17], much effort has been invested for assessing and improving software process capability and technology during the last decade. However, precise knowledge about the effects that specific process improvement actions have on specific customer defined product quality characteristics is still scarce.

In the recently completed European research project PROFES (PROduct Focused process improvement for Embedded Systems [32]), the relation between product and process characteristics has been investigated more closely. The PROFES improvement methodology [5, 31] integrates several well-proven improvement techniques, such as process assessment [21, 27] and goal-oriented measurement [10, 37], and promotes a systematic and iterative approach of continuous improvement according to the Quality Improvement Paradigm (QIP) [4]. The core element of the PROFES improvement methodology is the concept of so-called product-process dependency models (PPD models) [30]. The purpose of PPD models is to help limit process improvement activities to those development technologies and processes that are most effective with regards to achieving specific customer-defined product quality goals.

The PROFES improvement methodology provides guidelines for developing, using and evolving PPD models [31]. In order to achieve the product quality targets set by the customer, it is crucial that the PPD models be valid. More precisely speaking, validity of a PPD model means that in a given context the technology proposed for application in a particular development process significantly helps achieve a pre-defined product quality target. The most reliable way to validate PPD models is to generate empirical evidence from pilot applications [7], which is usually time-consuming and sometimes risky. Hence an interesting issue associated with the concept of modelling and exploiting product-process dependencies is the assessment of the actual effectiveness of not yet fully empirically validated PPD models.

When investigating a phenomenon of interest happens to be unfeasible in the real environment, or at least overly costly or risky, a common engineering prac-

tice consists of building a model that reproduces this phenomenon and which can be studied by simulation. The model is a mathematical abstraction that acts as a substitute for the real entities generating the phenomenon of interest but which is more amenable to manipulation. It is tempting to adopt the principles of modelling and simulation to analyse the (potential) effectiveness of PPD models in software development projects. In the remainder of this paper, a simulation-based approach will be proposed for experimenting with PPD models in a laboratory-like setting, before applying them in real software development projects.

The structure of the paper is as follows. In the next section, the basic concepts of PPD models are briefly presented (Sect. 2). Then, the simulation technique System Dynamics (SD) is introduced, and its suitability for analysing the effectiveness of proposed PPDs is motivated (Sect. 3). In Section 4, the approach for simulation-based experimentation with PPD models is outlined. Then, the approach is illustrated in a case study using a generic SD model and an example scenario for experimenting with two PPD models (Sect. 5). Section 6 discusses the results of the experiments conducted. Eventually, Section 7 summarises the results of the paper and gives an outlook to promising future research paths.

2 Product-Process Dependencies

PPD repositories are a core element of the PROFES improvement methodology. They contain an organised collection of so-called PPD models. A PPD describes the impact that a particular software engineering technology has on a certain software quality characteristic when applied in a certain development process in a specific project context.

The following sub-sections present the standard PPD model structure, the PPD model life-cycle, and a brief discussion of issues associated with PPD model validation.

2.1 PPD Model Structure

The generic structure of a PPD model consists of a main section and a context section (Fig. 1 shows an example).

PPD Model		
Product Quality	Maturity (e.g., measured in terms of defect density)	
Process	Software design specification	
Technology	Formal inspection (e.g., according to Fagan [14])	
Context Section		
Context Factor 1	Size of inspection team	1-2 **3-5** 5-10
<…>	<…>	<…>
Context Factor n	Size of inspected document	**small average** large very_large

Figure 1 Example PPD model (adopted from [26])

- Main section:
  - Slot 1 (Product Quality): The Product Quality slot specifies the product quality characteristic that is affected by the suggested process improvement activity. It is recommended to base the specification on a well-defined and accepted taxonomy, e.g., as provided by standard ISO9126 [18].
  - Slot 2 (Process): The Process slot specifies in which of the software processes the suggested process improvement activity takes place. Again, it is recommended to base the specification on a well-defined and commonly accepted taxonomy, such as standard process taxonomies for assessments (e.g., as provided by standard ISO15594 – also known as SPICE [19]).

– Slot 3 (Technology): The Technology slot specifies the method, technique, practice, or tool that the suggested process improvement activity applies. As for the other slots, it is recommended to base the specification on a well-defined and commonly accepted taxonomy. In this case, however, ready-to-use standards are not available, but company-specific taxonomies, or taxonomies published by organisations such as the Software Engineering Institute can be used as a starting point [36].

- Context section: The context section specifies the conditions under which the PPD model can be applied successfully. For each known context factor, a description and a range of possible values is provided. Which of the possible values are feasible is determined based on empirical evidence (i.e., from PPD model applications in software projects). The joint set of feasible values (marked by bold font in Fig. 1) determines the context in which the PPD model is recommended for use.

More detailed information about different types of PPD models can be found in [31].

2.2 PPD Model Life-Cycle

The PPD model life-cycle comprises three phases: development, usage, and evolution (see Fig. 2).

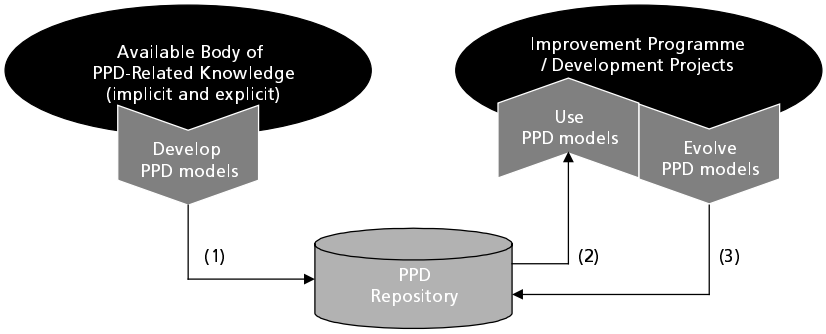

Figure 2 PPD model life cycle [6]

2.2.1 PPD Model Development

Developing PPD models is a combined analysis and design task. The objective is to identify, analyse, and package SPI relevant information such that it can be easily stored, reused, and maintained. The information contained in PPD models can be obtained by using different strategies and information sources, i.e. interviews with experienced software professionals, systematic measurement

programmes (e,g, following the GQM approach), and process assessments. For identifying generic, i.e. not company-specific PPDs, scientific literature and surveys can provide good baseline information.

2.2.2 PPD Model Usage

Within SPI programmes the most relevant usage areas of PPD models are the identification of potential process improvement actions, and the focusing of process assessments based on the previously defined product quality goals. The procedure for identifying improvement actions with the help of PPD model repositories (PPD repositories) is outlined in [26] and fully described in [31]. The most important steps of this procedure include:

- STEP 1: Identification of product quality goal.
- STEP 2: Identification of process(es) for which improvement actions are expected to be most beneficial. This step can be based on process assessment results and information derived from repositories of existing PPD models.
- STEP 3: Retrieve all PPD models that provide improvement suggestions for the product-process combination established in steps 1 and 2.
- STEP 4: Rank the PPD models with regards to their suitability for the next software development project. The ranking is based on the degree to which the context information provided in the PPD models match with the project characteristics.
- STEP 5: Select those technologies for introduction in the next software development project, which are suggested by the highest ranked PPD models.

2.2.3 PPD Model Evolution

Because the context of software development constantly evolves, enhancement of the PPD models is necessary. In each application of a PPD model new experience about the relationship between product quality and development processes is gained, hence the PPD models need to be updated and refined in order to reflect the evolved context and experience.

2.3 PPD Model Validation

The most problematic and difficult task is the validation of PPD models. The best way to validate a PPD model is to set up a related measurement programme, run a pilot application, and collect measurement data. Based on the analysis of the collected data, the validity can be judged (for an example cf. [8]). Since preparing and running pilot projects is usually rather costly – and sometimes risky – it may be wise to use simulation techniques for pre-checking

the plausibility of successful PPD model implementation before conducting the actual validation task.

Basically, three situations for simulation-based plausibility checking of PPD models can be identified:

1. Before initial empirical validation of a PPD model and its inclusion into the PPD repository: Plausibility checks based on simulations can indicate that chances of successful empirical validation are low. This is a useful information when deciding whether to conduct (expensive) pilot projects.
2. Before implementation of a technology in a real project, which is based on the recommendation of a PPD model taken from an existing PPD repository: Simulation results can help to reduce the risk of failure, particularly in situations when a complete matching of the context information with the actual project context cannot be achieved.
3. In order to further evaluate the actual implementation conditions (i.e., the context information) of existing PPD models before application in new projects. This can be particularly useful for generic PPD models, because context information for them is typically not very detailed.

In the following sections, more details are provided on how to support PPD model validation with the help of system dynamics simulation models.

3 System Dynamics Modelling

The potential of simulation in software engineering has been pointed out by many researchers during the last decade (e.g., cf. [12, 39]). Marc Kellner et al. [20] list several promising application areas for simulation-based analysis in software organisations, including: understanding, training and learning, planning, control and operational management, strategic management, process improvement and technology evaluation.

Among the most popular simulation modelling methods proposed by researchers (a selection is provided in [33]), System Dynamics (SD) seems to be a very promising approach, because it facilitates an integrated modelling of product, process, technology, and people. Originally, the SD modelling and simulation approach was developed by Jay Forrester at the MIT to tackle socio-economic and/or socio-technical problems [15, 16]. SD is based on the assumption of the ubiquity of feedback processes in human interactions: considered from a high level of abstraction, a socio-economic or socio-technical system can be modelled as a feedback structure, whose complex behaviour is generated by the interaction of many (possibly non-linear) loops over time.

In the late 1980s, researchers and practicioners have started to apply SD modelling to the field of software engineering. Published examples of SD applications in software development cover a variety of issues such as software project management [1, 13, 24], the impact of software process improvements on cycle-time [38], the impact of systems engineering on software development [28], concurrent software engineering [29], effects of software quality improvement activities [2, 25], software reliability management [35], software maintenance [11], and software evolution [23].

Due to the high flexibility of the SD modelling approach, in an SD model, the underlying cause-effect structure of a whole software development system (consisting of products, processes, technologies, and people) can be captured on an aggregated level, and translated into functional relationships formally represented by mathematical equations, which are then the basis for running computer simulations. The structure of the modelled system is graphically represented by the so-called flow diagram. The basic modelling constructs typically used in a SD flow diagram are depicted in Fig.3.

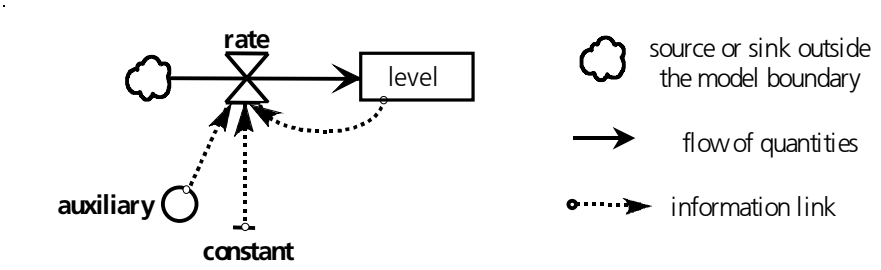

Figure 3 Schematic conventions of flow diagrams

Level variables (sometimes also referred to as state variables) describe the state of the system. They accumulate (or integrate) the results of action in the system, an action being always materialised by flows in transit. The derivative of a level, or equivalently the rapidity at which it is changing, depends on its input and output flows. The computation of a level is approximated by a difference equation of the following form:

$$Level(t+dt) = Level(t) + \left(\sum input\_rates - \sum output\_rates\right)dt \qquad (1)$$

The rates are what change the value of levels. Their equations define how available information is used in order to generate actions. A rate has four conceptual components: an observed condition is compared to a goal, and the discrepancy found is taken as the basis for action (flow) generation. The rate equation, which formalises this policy, is an algebraic expression that depends only on levels and constant values. Auxiliary variables can be used for intermediate computation.

4 Simulation-based Experimentation with PPD Models

The procedure that describes how to use SD models for experimenting with PPD models in order to check their plausibility is quite straightforward. Basically, it consists of three steps:

- STEP 1 (baseline SD model): Development of a SD model that captures the typical behaviour of software development projects in the organisation (with consideration of specific characteristics – if existing – of those development projects that are candidates for piloting the suggested improvement action). In order to be suitable for PPD model evaluation it is necessary that the model explicitly represent the product-process combination of interest. The related information can be obtained from the main section of the candidate PPD model. Note that details about the technology being applied are not yet implemented in the baseline SD model.
- STEP 2 (extended SD model): Extend the SD model such that it correctly captures the process changes implied by the implementation of the technologies suggested by the candidate PPD model. This extension often implies the implementation of new rate and level variables in the SD model. In order to produce useful simulation results, also the information contained in the context section of the PPD model has to be reflected adequately by the extended SD model. Adding new model parameters with information links affecting the control of rate variables typically does this.
- STEP 3 (evaluation scenario): Define a scenario for evaluating the effects of the proposed improvement action through simulation, perform the scenario, and analyse the results. Essentially, the scenario consists of running the baseline and extended SD models with carefully designed sets of parameter values.

Obviously, in order for this procedure to work, the validity of the SD model is crucial. Even though there exist tests for checking the validity of SD models [3], a problem is associated with STEP 2 of the procedure, because empirical validation of those model elements representing the effects of the PPD model is per definitionem impossible, and validity checking must rely on expert judgement alone. Clearly, expert judgement can only extend to checking the correct implementation of all effects the new technology proposed by the candidate PPD model is expected to have. Note that the added value of running simulations with a system dynamics model originates in the ability to visualise and analyse the system behaviour generated by the interaction of many interrelated cause-effect relations, where each individual cause-effect relationship is believed to hold if looked at it in isolation. In any case, given the difficulties associated with model validation, simulation results should mainly be used for refutation

of the candidate PPD model. That is, simulation results are particularly interesting when they suggest that the candidate PPD model is not as effective as expected, because this is a warning signal indicating a certain risk that – in the case that the simulation model was actually valid – empirical validation might not succeed, too.

In addition to checking the plausibility of the effectiveness of proposed technologies, experimentation with PPD models may have another benefit. A PPD model that has not yet been evaluated in real projects tends to be quite generic, i.e. its context section does not yet contain much detail. Simulation-based experimentation can also be used to explore the application context of such PPD models. In the following section, a (hypothetical) case study is conducted to illustrate this usage of the procedure outlined above.

5 Case Study

Starting point of the case study is the supposed need of a software organisation to improve the maturity of their software products. Product maturity is expressed in terms of (expected) defect density after system test. The case study involves three elements: the simulation model that is used for experimenting with candidate PPD models, a set of candidate PPD models, and the PPD model evaluation scenario. All three elements are briefly described in the following sections.

5.1 System Dynamics Simulation Model GEN-PROSIM

Starting from ideas similar to [1, 24, 25], Fraunhofer IESE developed the SD model GEN-PROSIM (GENeric PROject Simulation Model). The main purpose of this model is to demonstrate how SD models can be used for the management of software development projects, and for the improvement of the underlying processes and technologies, taking under consideration trade-off effects between time, cost, and quality, simultaneously.

The model structure represents in a simplified, generic waterfall-model like fashion the core phases of a typical software development project: Design, Implementation, Test. The calibration of the model was not based on a special case or on exhaustive empirical research, but on the functional relationships between effort, time, and size, as suggested by the well-known COCOMO model [9].

In total, GEN-PROSIM consists of five interrelated sub-models (views):

- Production: This view represents a typical software development cycle consisting of the following chain of transitions: set of requirements → design documents → code → tested code. Note that the detection of defects during testing only causes reworking of the code (and not of the design documents).
- Quality: In this view, the defect co-flow is modelled, i.e.: defect injection (into design or code) → defect propagation (from design to code) → defect detection (in the code during testing) → defect correction (only in the code).
- Effort: In this view, the total effort consumption for design development, code development, code testing, and defect correction (rework) is calculated.
- Initial Calculations: In this view, using the COCOMO equations, the normal value of the central process parameter "productivity" is calculated. The normal productivity varies with assumptions about the product development

mode (organic, semi-detached, embedded) and characteristics of the project resources available (e.g. developer skill).

– Productivity, Quality & Manpower Adjustment: In this view, project-specific process parameters, like (actual) productivity, defect generation, effectiveness of QA activities, etc., are determined based on a) planned target values for manpower, project duration, product quality, etc., and b) time pressure induced by unexpected rework or changes in the set of requirements.

The most important model parameters are listed in Table 1 according to their role for PPD model evaluation. It should be noted, however, that, in total, the model user can chose from more than 30 parameters in order to adapt the model to a specific environment.

Input Parameters		**Output Parameters**
Project Characterisation Parameters	**Project Management Parameters**	
Initial_job_size_in_tasks [functional units]		Job_size_in_tasks [implemented and tested functional units]
Project_complexity [organic, semi-detached, embedded]	Planned_manpower [persons] (optional)	Project_time [weeks] (project total and per phase)
Manpower_skill [low, average, high]	Planned_completion_time [weeks] (optional)	Effort [person weeks] (project total and per phase)
	Goal_field_defect_density [defects per implemented functional unit] (optional)	Field_defect_density [defects per implemented functional units after test]

Table 1 GEN-PROSIM model parameters

5.2 PPD Models Examined

Starting from the product quality improvement goal, a search in an available repository of PPD models revealed two quite generic PPD models as candidates for implementation (see Fig. 4). The first PPD model suggests implementing formal inspections during the coding phase, the second PPD model suggests implementing formal inspections during the design phase. In both cases, the context section is still quite short and no contextual constraints on the value ranges of the few defined context factors have been identified.

PPD Model A		
Product Quality		Defect density
Process		Code development
Technology		Formal inspection
Context Section		
CF-1	Project type	**Organic, semi-detached, embedded**
CF-2	Project size	**Small, average, large**
CF-3	Manpower skill	**Low, average, large**

PPD Model B		
Product Quality		Defect density
Process		Design development
Technology		Formal inspection
Context Section		
CF-1	Project type	**Organic, semi-detached, embedded**
CF-2	Project size	**Small, average, large**
CF-3	Manpower skill	**Low, average, large**

Figure 4 PPD models for coding (A) and design (B) phase

The integration of the PPD models into SD model GEN-PROSIM affected three views, namely Production, Quality, and Effort. The Production view must offer the possibility to conduct design and/or code inspections. The Quality view must account for early defect detection during design and/or code inspections. The Effort view must account for additional effort due to inspection activities and induced rework resulting from defect detection during inspections. Fig. 5 depicts the integration of the PPD models into the Production view of the GEN-PROSIM flow graph. Gray ovals mark the variables that were added to the model, i.e. des_insp_practice (the share of the design documents that undergo inspections), des_insp_rate (design inspection rate), impl_insp_practice (the share of code documents that undergo inspection), and impl_insp_rate (code inspection rate).

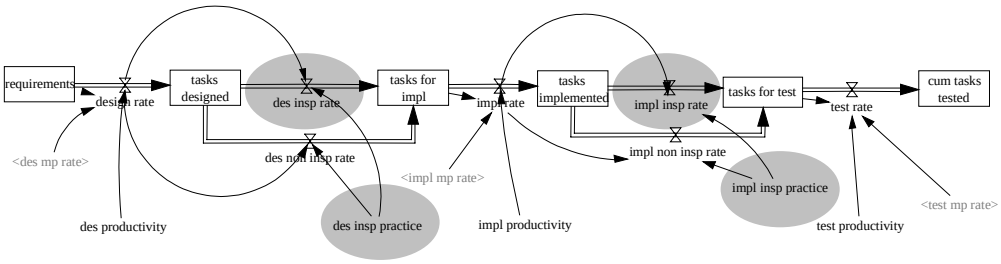

Figure 5 Production view of SD model GEN-PROSIM with PPD model implementation

5.3 PPD Model Evaluation Scenario and Simulation Results

With the help of the extended SD model GEN-PROSIM, the two suggested PPD models can be evaluated according to the following scenario:

- STEP 1 (baseline): Run simulation without inspections (des_insp_practice = 0, impl_insp_practice = 0) and store (estimated) defect density after testing (field_defect_density).
- STEP 2 (evaluation of PPD model A): Run simulation with code inspections (des_insp_practice = 0, impl_insp_practice = 1) and store (estimated) defect density at test end (field_defect_density).
- STEP 3 (evaluation of PPD model B): Run simulation with design inspections (des_insp_practice = 1, impl_insp_practice = 0) and store (estimated) defect density at test end (field_defect_density).
- STEP 4 (evaluation of PPD model A & B): Run simulation with code and design inspections (des_insp_practice = 1, impl_insp_practice = 1) and store (estimated) defect density at test end (field_defect_density).
- STEP 5: Compare results of steps 2-4 with step 1 (baseline) and draw conclusions.

Note that this scenario can be run for any combination of value assignments in the PPD model context sections. In Fig. 6 below, simulation results for the following context settings are presented: project type = semi-detached, project size = average (1000 functional size units), and manpower skill = average.

The results of the simulation-based analysis are shown in Fig. 6. The lines 1 to 4 show the trade-off relation between project duration and field defect density (FDD) for all four cases, i.e. baseline, with PPD model A, with PPD model B, with PPD models A and B. Note that manpower allocation was kept constant for all simulations, total effort and project duration are proportional, and thus effort numbers do not need to be considered. In order to simplify the graph, relative numbers are plotted on the x-axis, i.e. project_duration equals 1 for the baseline case (line 1). As can be seen, it is possible to impose variation on the model variables for project duration and FDD. This is achieved by setting a specific FDD target value. Depending on whether the target value is greater than or less than the typical value suggested by the model (indicated by 'x' on lines 2 to 4 in Fig. 6), testing is stopped earlier or later and thus less or more defects are detected.

In any case, when looking at Fig. 6, there is a clear ranking among the four situations. Related to the baseline case (field_defect_density = 1.79 defects per implemented functional unit), the application of code inspections (PPD model A), and design inspections (PPD model B) results in better product quality. In addition, the numbers show a clear ranking among results from STEP 2 (with PPD model A) and STEP 3 (with PPD model B), and in addition the data shows that the combined application of design and code inspections is even better wrt. to FDD.

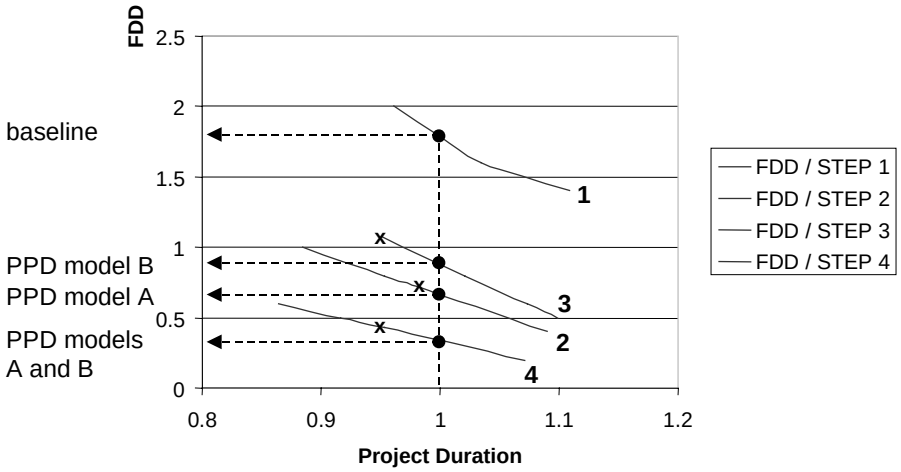

Figure 6 Simulation results (with rework effort relation 1: 2.5: 7.5)

A surprising detail of the simulation results is the fact that PPD model A ranked better than PPD model B. Why is the overall quality of the software better when conducting code inspections instead of design inspections? A closer look at how the defect co-flow was implemented in GEN-PROSIM uncovered that there is a relationship between the point in time of defect detection, and the associated average rework effort induced per defect. If, in a particular phase, the rework effort per defect increases (e.g., due to increased difficulty of analysis and correction), then there is less manpower left over for development activities. This creates schedule pressure, which in turn reduces the available effort for conducting inspections, and at the same time increases the probability of introducing new defects.

In Fig. 6, the relation of the average rework effort per defect between the phases design, coding, and testing was 1: 2.5: 7.5, i.e. the later a defect has been detected the more rework effort is induced for defect correction. If this relation is altered, say to 1: 7.5: 7.5, the ranking of PPD models A and B is reversed (cf. Fig. 7). This observation indicates that there is a new context factor – not yet listed in the context section of the PPD models (i.e., average rework effort per defect) – that should be further investigated and eventually included in the context sections of the PPD models.

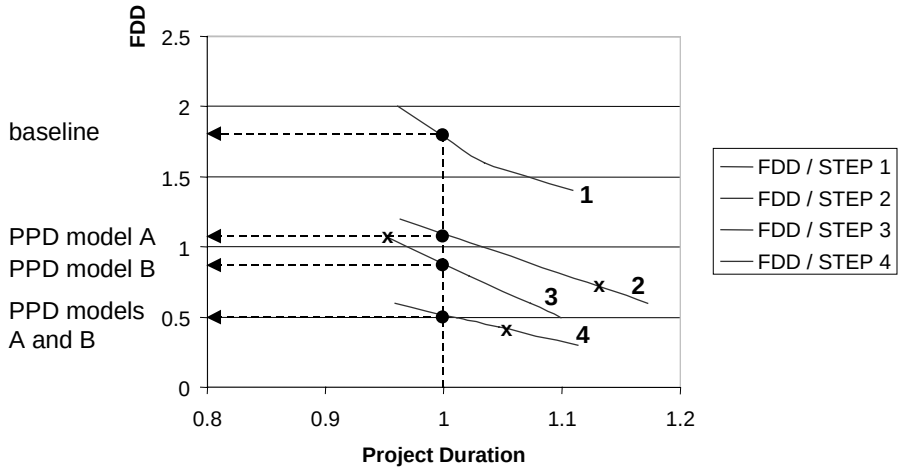

Figure 7 Simulation results (with rework effort relation 1: 7.5: 7.5)

6 Discussion

The case study conducted in Section 5, sheds light on one possibility of applying the overall approach of simulation-based experimentation with PPD models. In the presented case, the purpose of the experimentation was to explore the (potential) effectiveness of two PPD models in a generic setting. It could be demonstrated that systematic exploration is feasible, and that, in addition, a further refinement of the context section of the PPD models could be triggered based on the simulation results. Of course, due to the lack of empirical validity of the SD model used, all numbers generated through simulation are only of limited value, and should only be interpreted qualitatively. In particular, the simulation results cannot be used for checking the plausibility of the candidate PPD models with respect to the effectiveness of the proposed technologies in a real development project. If this was the goal, much more effort had to be put in developing a valid SD model that is calibrated to a real development environment. Guidance for doing this can be found in [22, 34].

It should be noted, however, that even when working with generic (and/or actually invalid) models, running simulations is a useful tool for deepening the experts' understanding of software development projects and SPI actions. Particularly when the simulation results do not match the expectations of the experts involved, both the simulation model as well as the mental models of the experts will be subject to double-checking and possibly revision. This will in any case trigger learning about reality, and the perception of reality, with substantial positive effects on the management of software projects and SPI programmes.

7 Conclusion and Future Work

The case study conducted in Section 5 has demonstrated that the suggested approach of simulation-based experimentation with PPD models is feasible and useful. More work, however, is needed to mature the overall approach in order to make it a reliable, cheap, and easy-to-apply support tool for decision makers in SPI programmes. To achieve this, future research will focus on three areas:

1. Provision of systematic guidance for conducting simulation-based explorative experimentation with generic PPD models.
2. Provision of systematic guidance for conducting simulation-based plausibility checks on yet mature PPD models before implementation of the proposed technology in an upcoming development project.
3. Deeper investigation of the relationships and synergies between SD modelling and PPD modelling in order to develop a framework for model-based learning in software organisations.

References

1. Abdel-Hamid, T. K., Madnick, S. E.: Software Projects Dynamics – an Integrated Approach. Prentice-Hall (1991)
2. Aranda, R. R., Fiddaman, T., Oliva, R.: Quality Microworlds: modeling the impact of quality initiatives over the software product life cycle. American Programmer (May 1993) 52-61
3. Barlas, Y.: Multiple Tests for Validation of System Dynamics Type of Simulation Models. European Journal of Operational Research 42 (1989) 59-87
4. Basili, V. R., Caldiera, G.: Improve Software Quality by Reusing Knowledge and Experience. Sloan Management Review (Fall 1995) 55-64
5. Birk, A., Järvinen, J., Komi-Sirviö, S., Kuvaja, P., Oivo, M., Pfahl, D.: PROFES – A Product Driven Process Improvement Methodology. In: Proceedings of the European Conference on Software Process Improvement (SPI'98), Monte Carlo, December 1 – 4 (1998)
6. Birk, A., Järvinen, J., Oivo, M., Pfahl, D.: Product-Driven Process Improvement Using the PROFES Improvement Methodology. Tutorial presented at 11th International Conference on Software Engineering and Knowledge Engineering (SEKE'99), Kaiserslautern, June 16 (1999)
7. Birk, A., Järvinen, J., Solingen, R. van: A Validation Approach for Product-Focused Process Improvement. In: Proceedings of First International Conference on Product Focused Software Process Improvement (Profes´99), VTT Symposium Series 195, VTT Technical Research Centre of Finland, Espoo, Finland (1999) 29 – 48
8. Birk, A., Solingen, R. van, Järvinen, J.: Business Impact, Benefit, and Cost of Applying GQM in Industry: An In-Depth, Long-Term Investigation at Schlumberger RPS. In: Proceedings of the Fifth International Symposium on Software Metrics (Metrics´98), Bethesda, Maryland, November 19 – 21 (1998)
9. Boehm, B. W.: Software Engineering Economics. Prentice-Hall (1981)
10. Briand, L. C., Differding, C., Rombach, H. D.: Practical Guidelines for Measurement-Based Process Improvement. Software Process Improvement and Practice 2(4) (1996) 253-280
11. Cartwright M., Shepperd, M.: On building dynamic models of maintenance behaviour. In: Kusters, R., Cowderoy, A., Heemstra, F., and

Veenendaal, E. van (eds.): Project Control for Software Quality. Shaker Publishing (1999)

12. Christie, A. M.: Simulation: An Enabling Technology in Software Engineering. CROSSTALK – The Journal of Defense Software Engineering (April 1999) 2-7
13. Cooper, K. G., Mullen, T.: Swords and Ploughshares: the Rework Cycles of Defence and Commercial Software Development Projects. American Programmer 6(5) (1993) 41-51
14. Fagan, M.: Design and Code Inspections to Reduce Errors in Program Development. IBM Systems Journal 15(3) (1976) 182-211
15. Forrester, J. W.: Industrial Dynamics. Productivity Press (1961)
16. Forrester, J. W.: Principles of Systems. Productivity Press (1971)
17. Humphrey, W. S.: Managing the Software Process. Addison-Wesley Publishing Company (1990)
18. ISO/IEC 9126 Standard. Information technology – Software product evaluation – Quality characteristics and guidelines for their use. International Organisation for Standardisation (Ed.), Case Postale 56, CH-1211 Geneva, Switzerland, first edition 15 December (1991)
19. ISO/IEC TR 15504 Standard. Information technology – Software process assessment – Part 1-9. Technical Report type 2, International Organisation for Standardisation (Ed.), Case Postale 56, CH-1211 Geneva, Switzerland (1998)
20. Kellner, M. I., Madachy, R. J., Raffo, D. M.: Software process simulation modeling: Why? What? How? Journal of Systems and Software 46(2/3) (1999) 91-105
21. Kuvaja, P., Similä, J., Krzanik, L., Bicego, A., Saukkonen, S., Koch, G.: Software Process Assessment & Improvement – The BOOTSTRAP Approach. Blackwell Publishers (1994)
22. Lebsanft, K., Pfahl, D.: Knowledge Acquisition for Building System Dynamics Simulation Models: An Experience Report from Software Industry. In: Proceedings of the 11th Int'l Conference on Software and Knowledge Engineering (SEKE), Kaiserslautern (June 1999) 378-387
23. Lehman M. M., Ramil, J. F.: The impact of feedback in the global software process. Journal of Systems and Software 46(2/3) (1999) 123-134
24. Lin, C. Y., Abdel-Hamid, T., Sherif, J. S.: Software-Engineering Process Simulation Model (SEPS). Journal of Systems and Software 38 (1997) 263-277
25. Madachy, R. J.: System Dynamics Modeling of an Inspection-Based Process. In: Proceedings of the 18th International Conference on Software

Engineering (ICSE'96), Berlin, Germany, IEEE Computer Society Press (March 1996)

26. Oivo, M., Birk, A., Komi-Sirviö, S., Kuvaja, P., Solingen, R. van: Establishing Product Process Dependencies in SPI. In: Proceedings of European Software Engineering Process Group Conference (European SEPG'99), Amsterdam, The Netherlands, 7 – 10 June (1999)
27. Paulk, M. C., Curtis, B., Chrissis, M. B., Weber, C. V.: Capability Maturity Model, Version 1.1. IEEE Software, (July 1993), 18-27
28. Pfahl, D., Lebsanft, K.: Using Simulation to Analyse the Impact of Software Requirement Volatility on Project Performance. To appear in the Proceedings of the European Software Control and Metrics Conference (ESCOM), Munich, 17-20 April (2000)
29. Powell, A., Mander, K., Brown, D.: Strategies for lifecycle concurrency and iteration: A system dynamics approach. Journal of Systems and Software 46(2/3) (1999) 151-162
30. PROFES PPD Repository at URL http://www.iese.fhg.de/projects/profes/PPDRepository
31. PROFES User Manual, can be downloaded from URL http://www.profes.org
32. PROFES web site at URL http://www.profes.org
33. Raffo, D. M., Harrison, W., Kellner, M. I., Madachy, R. J, Martin, R., Scacci, W., Wernick, P. (eds.): Special Issue on: Software Process Simulation Modeling. Journal of Systems and Software 46(2/3) (1999)
34. Richardson, G. P., Pugh, A. L.: Introduction to System Dynamics Modeling with DYNAMO. Productivity Press, Cambridge (1981)
35. Rus, I., Collofello, J., Lakey, P.: Software process simulation for reliability management, Journal of Systems and Software 46(2/3) (1999) 173-182
36. Software Engineering Institute. C4 Software Technology Reference Guide – A Prototype. Handbook CMU/SEI-97-HB-001, Software Engineering Institute (1997)
37. Solingen, R. van, Berghout, E.: The Goal/Question/Metric method: A practical guide for quality improvement of software development. McGraw-Hill Publishers (1999)
38. Tvedt, J. D., Collofello, J. S.: Evaluating the Effectiveness of Process Improvements on Development Cycle Time via System Dynamics Modeling. In: Proceedings of the Computer Science and Application Conference (COMPSAC) (1995) 318-325
39. Waeselynck, H., Pfahl, D.: System Dynamics Applied to the Modelling of Software Projects. Software Concepts and Tools 15(4) (1994) 162-176

Document Information

Title:	Using Simulation to Visualise and Analyse Product-Process Dependencies in Software Development Projects
Date:	February 2000
Report:	IESE-013.00/E
Status:	Final
Distribution:	Public