Fraunhofer Institut
Experimentelles
Software Engineering

Using Simulation to Analyse the Impact of Software Requirement Volatility on Project Performance

Authors:
Dietmar Pfahl
Karl Lebsanft[1]

[1] Siemens AG, ZT SE 3, Munich, Germany

Submitted for publication in
Proceedings of the ESCOM'2000

IESE-Report No. 003.00/E
Version 1.0
January 25, 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

During the last decade, software process simulation has been used to address a variety of management issues and questions. These include: understanding; training and learning; planning; control and operational management; strategic management; process improvement and technology adoption.

This paper presents a simulation model that was developed by Fraunhofer IESE for Siemens Corporate Technology. The purpose of this simulation model was to demonstrate the impact of unstable software requirements on project duration and effort, and to analyse how much money should be invested in stabilising software requirements in order to achieve optimal cost effectiveness.

The paper reports in detail on the various steps of model building, discusses all major design decisions taken, describes the structure of the final simulation model, and presents the most interesting simulation results of a case study.

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 programmes. Often, these improvement programmes are based on the recommendations received from regularly conducted software process assessments [1-2]. However, whether the suggested improvement actions are actually (cost-) effective in a given software development environment is usually hard to say. This is due to the high dynamic complexity of software development, which takes place in environments that are determined by products, processes, methods, techniques, technologies, tools, and people (customers, managers, project leaders, developers, etc.). All these entities interrelate through a network of dependencies, thus forming a software development system. The adequate approach for assessing (cost-) effectiveness of improvement actions would be to set up a measurement program, to conduct pilot projects, and to evaluate the impact of the improvement effort based on the analysis of empirical data [3-5].

Unfortunately, in large-scale industrial software production environments it is very time-consuming and costly to experiment with alternative development technologies (and processes) on real projects. When experimentation on the real system happens to be unfeasible, a common engineering practice consists of building a model that can be studied by simulation. The method System Dynamics, originated by Forrester at MIT during the 1950s to analyse the behaviour of socio-economic systems [6-7], provides the means for this kind of simulation-based analysis. System Dynamics (SD) modelling is based on expert knowledge elicitation and (if available) empirical data [8]. An SD model captures the underlying cause-effect structure of a software development system and translates it into functional relationships formally represented by mathematical equations, which are then the basis for running computer simulations that can be used as a first plausibility check on the (cost-) effectiveness of suggested improvement actions.

In a recent survey [9], several promising application areas for simulation-based analysis in software organisations have been listed, including: understanding, training and learning, planning, control and operational management, strategic management, process improvement and technology evaluation. Published examples of SD applications in software development cover a variety of issues

such as software project management [10-12], the impact of process improvements on cycle-time [13], concurrent software engineering [14], effects of software quality improvement activities [15-16], software reliability management [17], software maintenance [18], and software evolution [19].

This paper presents a simulation model that was developed by Fraunhofer IESE for Siemens Corporate Technology (Siemens CT). The purpose of this simulation model was a) to demonstrate the impact of unstable software requirements on project duration and effort, and b) to analyse how much effort should be invested in stabilising software requirements in order to achieve optimal cost effectiveness.

The paper provides background information on the reasons for conducting the study presented (Section 2), and then reports in detail on the various steps of model building (Section 3), discusses all major design decisions taken (Section 4), describes the structure of the final simulation model (Section 5), sketches the process of model calibration and validation (Section 6), and presents the most interesting simulation results of the case study conducted (Section 7). Based on the lessons learned from the case study, conclusions are drawn about the suitability of System Dynamics simulation models for analysing software development projects and processes (Section 8).

2 Background and motivation

The starting point for developing the simulation model was a CMM-compatible software process assessment [20-22], which Siemens CT had conducted within a Siemens Business Unit (Siemens BU). Usually, the main result of a software process assessment is a list of suggested changes to the software processes. In this case, the assessors' observations indicated that the software development activities were strongly affected by software requirement volatility. Moreover, due to the type of products developed by Siemens BU, i.e. products consisting of hardware (e.g. micro-controllers) and embedded software, the definition of software requirements was under direct control of systems engineering, and thus not totally under responsibility of the software department. During the assessment, the assessors observed that many system requirements that had already been addressed by software development were changed by the customer, or replaced by new requirements defined by systems engineering late in the project. In addition, there were many cases where system requirements that originally had been passed to software development eventually were realised by hardware, and vice versa. Based on these observations, the assessors expected that improvement suggestions that exclusively focused on software development processes (e.g., introduction of software design or code inspections) would not help stabilise software requirements. Since the software department that had ordered the process assessment strongly requested improvement suggestions that could be implemented under their responsibility, there was a need to find means that helped convince decision makers that first systems engineering had to be improved before improvements in software development could become effective. Hence the decision was made to charge Fraunhofer IESE with developing a simulation model that clarified the situation, and that investigated the cost-effectiveness of improvements in systems engineering with regards to software development.

3 Model building

The simulation model was developed using the System Dynamics method. A detailed model building process previously defined by Fraunhofer IESE guided the modelling activities [8].

The model building process was highly iterative. 13 increments were needed to come up with a base model that was able to capture the software development behaviour mode of interest, and which contained all relevant factors governing observed project behaviour. After two additional iterations, the simulation model was ready to be used for its defined purpose.

In total, 5 persons were involved in model building (4 persons at Siemens CT, 1 person at Fraunhofer IESE). Overall, model building and documentation consumed less than 2 person months of effort.

4 Design decisions

Besides the definition of the model boundaries and model granularity, the most important design decisions were related a) to the typical observable dynamics ("reference mode") of development projects at Siemens BU that the model should be able to reproduce through simulation, and b) to the assumptions about the most significant cause-effect relationships ("base mechanisms") governing the observed project dynamics.

4.1 Reference mode

The reference (behaviour) mode was determined by the dynamics of product evolution (i.e. a product is developed in three increments) and the dynamics of requirements generation (i.e. each product increment implements certain types of requirements) that typically can be observed during project performance (cf. Figure 1).

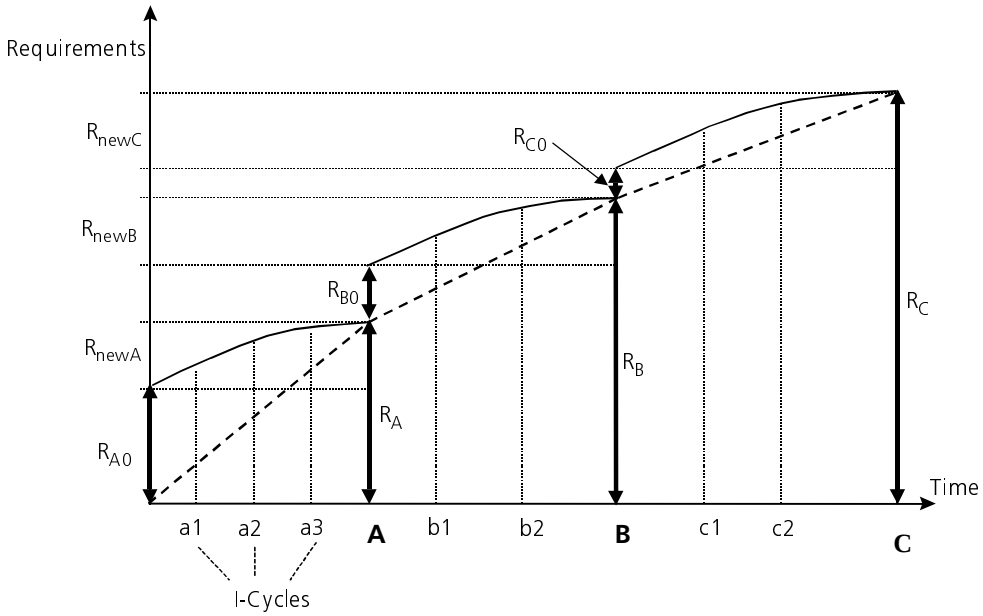

Figure 1: Typical pattern of product evolution during project performance.

4.1.1 Dynamics of product evolution

The growth of the software product is shown with a dashed line in Figure 1. The development of the software product is done in three subsequent, approximately equally long periods. During each period one increment is developed. The contents of the respective increments can be characterised as follows:

- Increment A: implements the base functionality (prototype)
- Increment B: implements all important requirements
- Increment C: implements all requirements (incl. customer-specific adaptations)

During the development of an increment, usually several releases that are subject to customer examination are created. The development cycles that are needed to create a release are called improvement cycles (I-Cycles).

4.1.2 Dynamics of requirements generation

At the beginning of each development period of an increment, a fixed set of requirements to start with is known (R_{A0}, R_{B0}, R_{C0}). During the development of an increment, new requirements are usually received from the customer (mostly as a result from the examination of releases). Typically, the number of new requirements shows a ceiling effect. More formally, the following properties can be observed:

- Number of software requirements at project start: R_{A0}
- Cumulated number of requirements for increment A: $R_A = R_{A0} + R_{newA}$
- Cumulated number of requirements for increment B: $R_B = R_A + R_{B0} + R_{newB}$
- Cumulated number of requirements for increment C: $R_C = R_B + R_{C0} + R_{newC}$
- Relationship between number of new requirements at start of an increment development: $R_{A0} > R_{B0} > R_{C0}$

It should be noted that only those requirements are shown in Figure 1 that actually will be contained in the final product. Alterations in the total number of requirements due to modification or replacement of requirements are not shown.

4.2 Base Mechanisms

For building the SD model it is necessary to identify the most important causal relationships that are supposed to generate the typical project behaviour. Based

on the insights that the Siemens CT experts gained during process assessment, the following six base mechanisms were identified:

- The average productivity of the workforce, measured as the number of implemented requirements per effort unit, is constant during the development of a product increment I_j (j = A, B, C). Between increments the following relations hold: prod (I_A) > prod (I_B) > prod (I_C).
- If requirements that have already been implemented are replaced by new requirements, rework cycles have to be conducted.
- The more rework cycles occur, the smaller is the average development productivity of the related increment. This affects effort consumption and project duration in addition to the fact that more requirements have to be developed than originally planned.
- Unstable definition of software requirements increases the number of rework cycles.
- Stability of software requirements definition is a measure of systems engineering quality. Systems engineering quality can be increased, if effort is invested for improvement actions.
- Generally, holding the project deadline has highest priority, i.e. if the project schedule is at risk, more manpower will be added to the project.

5 Model structure

The simulation model was implemented in a modular way using the System Dynamics tool Vensim® [23]. The main module represents the software development with its interface to systems engineering from which the software requirements are received. Four additional modules describe certain aspects of software development in more detail, namely: manpower allocation and productivity, effort and cost calculation, generation of new software requirements, and control of incremental software development. The following sub-sections provide rough descriptions of each model module.

5.1 Module 1: software development

The module "software development" represents all relevant elements of the software development process and its interface to systems engineering. In systems engineering, all customer requirements are collected and analysed. Those requirements that shall be implemented in the software product are filtered out and passed over to the software development process. Generally, there are three types of software requirements: those that are known at project start, those that are newly received in addition to the existing requirements during project performance, and those that are newly received in order to replace existing (and already implemented) requirements during project performance. Replacing requirements can be received at any time during project performance. The number of replacing requirements per period (e.g. per week) is proportional to the number of requirements known at project start. The factor that determines the number of replacing requirements (variable weekly_ replace _ factor) depends on the quality of systems engineering, i.e. the lower the systems engineering quality the greater the proportionality factor (cf. Section 5.3).

At project end, all software requirements are implemented in the software product. The speed with which a certain number of requirements can be implemented depends on the size of the workforce and the average productivity. Since the software product is implemented in three increments, and each increment is different in nature, there is a dedicated level of average productivity assigned to each increment. Typically, the productivity is such that the development periods of the increments are equally long. The average productivity is affected by the variable weekly_replace_factor, i.e. the greater the proportionality factor, the lower is the average productivity. This dependency relationship is justified by the observation that an increase in requirement replacements increases the amount of rework (represented by an increased number of I-Cycles).

5.2 Module 2: workforce allocation and adjustment

The module "workforce allocation and adjustment" represents all relevant elements of the software development process relevant for allocating software developers and adjusting their number during project performance (if necessary). The initial number of software developers is calculated based on the number of initial requirements according to the typical manpower allocation pattern at Siemens BU. Workforce adjustments during project performance become necessary when continuously calculated projections of the probable project termination (which are based on the current workforce size, the current development productivity, and the number of remaining requirements to be implemented) significantly differ from the planned overall project duration. According to nature and extent of the divergence, developers are added or taken away from the team, the adjustment being subject to realistic delay.

5.3 Module 3: effort and cost calculations

Based on the actual project duration and the workforce allocation, the module "effort and cost calculation" calculates the overall effort used to develop the software product. In parallel, the development cost is calculated by multiplication with a cost factor.

This module also determines the effort for conducting the systems engineering task using the variable effort_provided_for_systems_engineering as a policy parameter when running simulations. It is assumed that the quality of the systems engineering, and thus the stability of the requirements (expressed through the variable weekly_replace_factor) is a direct function of the effort invested. The assumed relationship between effort provided for systems engineering and the variable weekly_replace_factor is shown in Figure 2 below.

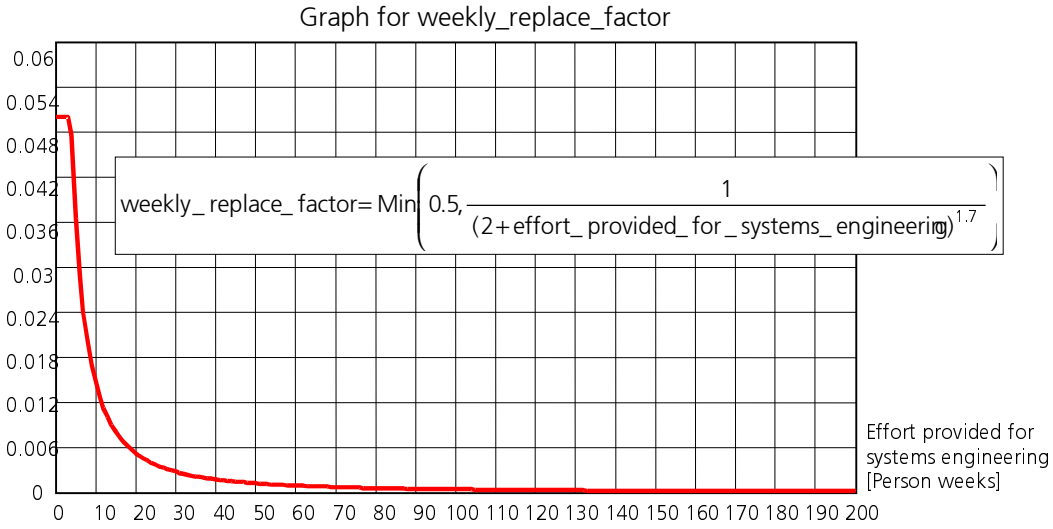

Figure 2: Relation between variable weekly_replace_factor and systems engineering effort.

5.4 Module 4: new requirements generation

The module "new requirements generation" determines the number of new requirements received during the development of increments A, B, and C (R_{newA}, $R_{B0} + R_{newB}$, $R_{C0} + R_{newC}$). The calculations are based on typical patterns observed at Siemens BU. Figure 3 shows the behaviour of the new requirements generation rate for increment A, adjusted to an initial number of requirements of 1000 (R_{A0}).

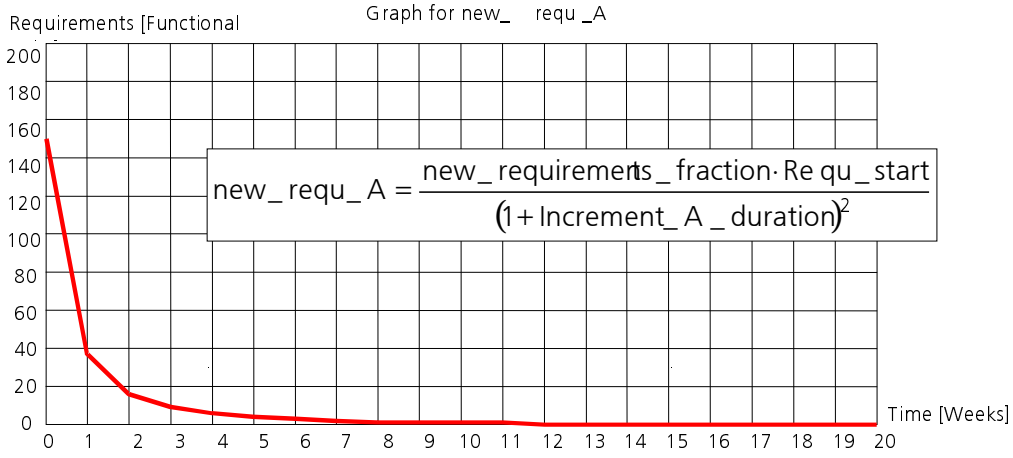

Figure 3: Relation between variable new_requ_A and time.

5.5 Module 5: co-ordination of increments

The module "co-ordination of increments" is needed for synchronising the model calculations related to the development of the respective software increments.

6 Model calibration and validation

The model was calibrated based on the knowledge of Siemens CT about the behaviour patterns of typical development projects at Siemens BU (baseline). Siemens CT gained their knowledge mainly through the process assessment previously conducted within the software organisation of Siemens BU. It should be noted that most of the information about software projects at Siemens BU was qualitative of nature (with the exception of effort data). Therefore, only relations between major variables were used to calibrate the model. Based on these relations, a normalised baseline project was defined through the model constants listed in Table 1.

Table 1: Model constants used for calibration.

Description	**Unit**	**Value**
Number of requirements at project start (R_{A0})	functional unit	1000
New requirements fraction (needed to calculate R_{newA}, R_{newB}, R_{newC})	dimensionless	0.15
Initial requirements fraction for increment B (needed to calculate R_{B0})	dimensionless	1.8
Initial requirements fraction for increment C (needed to calculate R_{C0})	dimensionless	0.5
Target project completion time	week	100
Nominal average productivity for increment A	functional unit / person week	11
Nominal average productivity for increment B	functional unit / person week	4
Nominal average productivity for increment C	functional unit / person week	2.5
Cost per effort unit	money unit / person week	2000

Model validation was mainly based on plausibility checks conducted by Siemens CT experts. In addition, an important necessary condition for model validity, i.e. the ability to reproduce the reference mode, was strictly fulfilled (for details on techniques for model validation cf. [24-25]). The patterns of software product growth (variable: SW_product) and generation of actually implemented requirements (variable: actual_all_SW_requirements) as produced by the simulation model are shown in Figure 4 for two different values of the policy variable effort_provided_for_systems_engineering (simulation runs n1 and n2). Note that the number of replaced requirements (variable: SW_replace_requ) and thus the total number of received requirements during project performance (variable: all_SW_requirements) varies as a result of the variation of the effort invested in systems engineering.

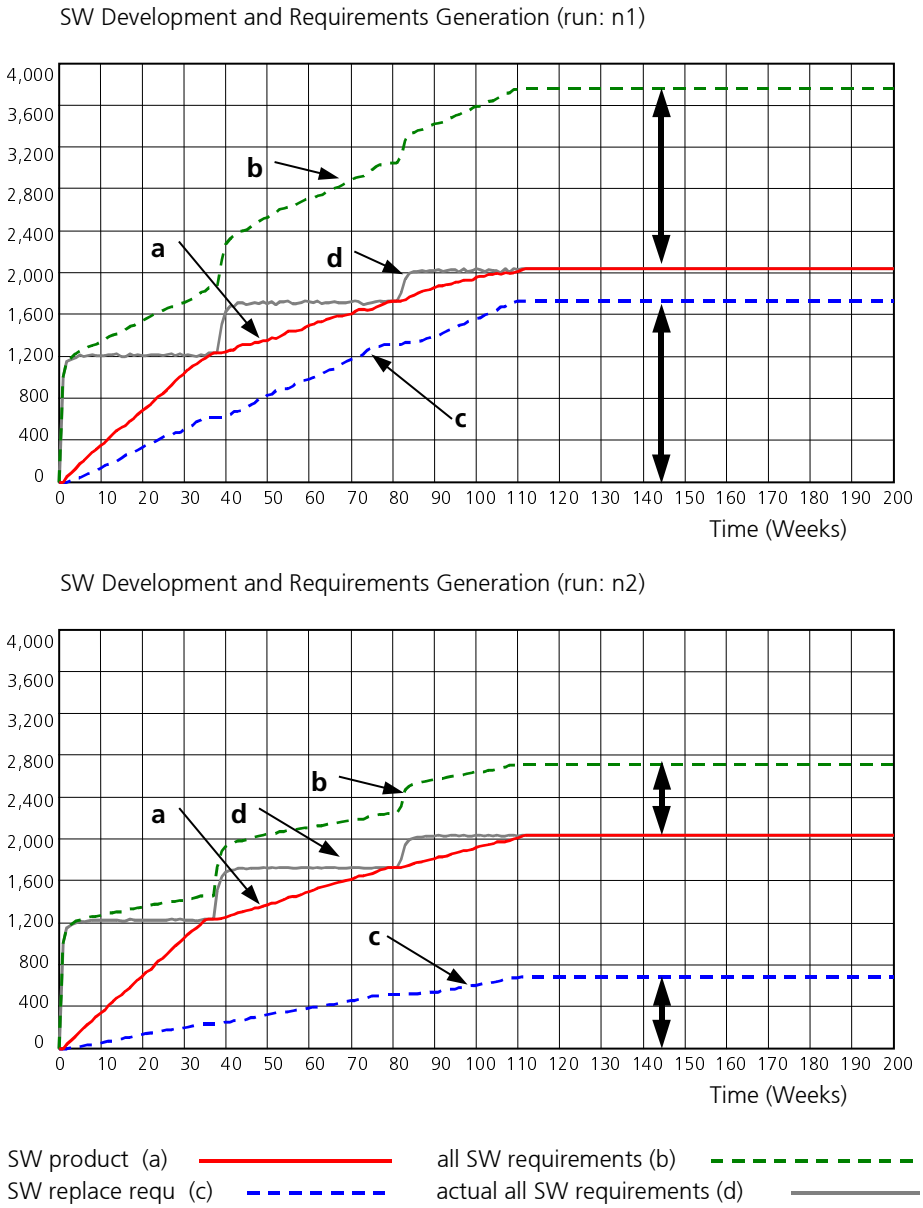

Figure 4: Reproduction of the reference behaviour (a: SW product – d: SW requirements).

7 Case study results

The question that had to be answered with help of the simulation model was: "How much effort should be invested into systems engineering in order to improve (software) requirements analysis and thus minimise the overall software development cost?" To answer this question, an equivalent mathematical minimisation problem was formulated:

$$total\_effort = x + \sum_{t=1}^{T} y(t) \longrightarrow min$$

with:
t: elapsed time (weeks)
T: project termination (weeks)
x: effort for systems engineering (person weeks)
y: weekly effort consumption for software development (person weeks / week)

The solution to this problem was found through variation of the policy variable x (model parameter: effort_provided_for_systems_engineering) and by using the built-in optimisation function of the simulation tool Vensim®, which applies the Fletcher-Powell algorithm [26]. The most important results are summarised in Table 2.

Table 2: Summary of simulation results.

Simulation run	n1	n2 (baseline)	n3	n5	optimal	n6
Syst. Eng. effort [person weeks]	5	10	15	30	42	50
SW dev. effort [person weeks]	875	586	499	452	420	416
Total effort [person weeks]	**880**	**596**	**514**	**482**	**462**	**466**
Cost-effectiveness	-	0	0.138	0.191	0.225	0.218
AARR per week [%]	1.83	0.73	0.40	0.14	0.08	0.06

It turned out, that an increase of the systems engineering effort share from 1.7% of the total effort (baseline situation) to 9.1% of the total effort (optimal situation) will reduce the overall cost for systems engineering and software development by more than 20% (from 596 to 462 person weeks). This effect is mainly due to the stabilisation of requirements, which is expressed in terms of the actual average requirements replacement (AARR) per week. In the optimal case, on average only 0.08% of the currently known (and yet implemented) requirements were replaced per week, adding up to a total of 29 replaced requirements during project performance.

8 Conclusions

Based on the simulations it was possible to demonstrate that software requirements volatility is extremely effort consuming for the software development organisation and that investments in systems engineering in order to stabilise requirements definition would well pay off. Of course, it must be pointed out that all results produced by the simulation model are based on qualitatively formulated assumptions underlying the model structure. Without thorough review of the model structure by experts of Siemens BU, and without a calibration of the model parameters and model functions to empirical data, the model cannot be used for precise point estimates in the sense of a predictive model.

However, having such a simulation model at hand makes it quite easy to visualise the critical project behaviour and to discuss the assumptions about the cause-effect relationships that are supposed to be responsible for the generated behaviour. In that sense, experts at Siemens CT felt that building the SD model was a useful exercise, and that similar models can help them in future process improvement projects with Siemens business units.

9 References

[1] Paulk, M. C., Curtis, B., Chrissis, M. B., and Weber, C. V., "Capability Maturity Model, Version 1.1", IEEE Software, July 1993, pp. 18-27.

[2] Kuvaja, P., Similä, J., Krzanik, L., Bicego, A., Saukkonen, S., and Koch, G., "Software Process Assessment & Improvement – The BOOTSTRAP Approach", Blackwell Publishers, 1994

[3] Basili, V.R., Caldiera G., "Improve Software Quality by Reusing Knowledge and Experience", Sloan Management Review, Fall 1995, pp. 55-64.

[4] van Solingen, R., Berghout, E., "The Goal/Question/Metric method: A practical guide for quality improvement of software development", McGraw-Hill Publishers, 1999.

[5] Birk, A., Järvinen, J., Komi-Sirviö, S., Kuvaja, P., Oivo, M., Pfahl, D., "PROFES - A product driven process improvement methodology", Proceedings of the European Conference on Software Process Improvement (SPI), Monaco, 1 - 4 Dec. 1998.

[6] Forrester, J.W., "Industrial Dynamics", Productivity Press, 1961.

[7] Forrester, J.W., "Principles of Systems", Productivity Press, 1971.

[8] Lebsanft K, Pfahl D, "Knowledge Acquisition for Building System Dynamics Simulation Models: An Experience Report from Software Industry". Proceedings of the 11th Int'l Conference on Software and Knowledge Engineering (SEKE), Kaiserslautern, June 1999, pp. 378-387.

[9] 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, pp. 91-105.

[10] Abdel-Hamid, T.K., Madnick, S.E., "Software Projects Dynamics – an Integrated Approach", Prentice-Hall, 1991.

[11] Lin, C. Y., Abdel-Hamid, T. and Sherif, J. S., "Software-Engineering Process Simulation Model (SEPS)", Journal of Systems and Software 38, 1997, pp. 263-277.

[12] Cooper, K.G. and Mullen, T., "Swords and Ploughshares: the Rework Cycles of Defence and Commercial Software Development Projects", American Programmer 6(5), 1993, pp. 41-51.

[13] Tvedt, J.D., Collofello, J.S., "Evaluating the Effectiveness of Process Improvements on Development Cycle Time via System Dynamics Modeling". Proceedings of the Computer Science and Application Conference (COMPSAC), 1995, pp. 318-325.

[14] Powell, A., Mander, K., and Brown, D., “Strategies for lifecycle concurrency and iteration: A system dynamics approach”, Journal of Systems and Software 46(2/3), 1999, pp. 151-162.

[15] Aranda, R.R., Fiddaman, T., and Oliva, R., "Quality Microworlds: modeling the impact of quality initiatives over the software product life cycle", American Programmer, May 1993, pp. 52-61.

[16] Madachy, R., "System Dynamics Modeling of an Inspection-Based Process", Proceedings of the 18th International Conference on Software Engineering (ICSE), Berlin, Germany, IEEE Computer Society Press, March 1996.

[17] Rus, I., Collofello, J., and Lakey, P., “Software process simulation for reliability management”, Journal of Systems and Software 46(2/3), 1999, pp. 173-182.

[18] Cartwright, M., Shepperd, M., “On building dynamic models of maintenance behaviour”, in Project Control for Software Quality (Kusters, R., Cowderoy, A., Heemstra, F., and van Veenendaal, E., eds.), Shaker Publishing, 1999.

[19] Lehman, M.M. and Ramil, J.F., “The impact of feedback in the global software process”, Journal of Systems and Software 46(2/3), 1999, pp. 123-134.

[20] Völker, A., “Software Process Assessments at Siemens as a Basis for Process Improvement in Industry”, Proceedings of the ISCN, Dublin, Ireland, 1994.

[21] Mehner, T., Messer, T., Paul, P., Paulisch, F., Schless, P., Völker, A., "Siemens Process Assessment and Improvement Approaches: Experiences and Benefits", Proceedings of the 22nd Computer Software and Applications Conference (COMPSAC), Vienna, 1998.

[22] Lebsanft, K., “Das Siemens Process Assessment”, in Evaluation und Evaluationsforschung in der Wirtschaftsinformatik (Heinrich, L.J. and Häntschel, I., eds.), Oldenbourg Verlag, 2000, pp.175-188.

[23] “Ventana Simulation Environment (Vensim®) - Reference Manual, Version 3.0”, Ventana Systems, Inc., 1997.

[24] Richardson, G.P. and Pugh, A.L., “Introduction to System Dynamics Modeling with DYNAMO”, Productivity Press, Cambridge, 1981.

[25] Barlas, Y., "Multiple Tests for Validation of System Dynamics Type of Simulation Models", European Journal of Operational Research, Vol. 42, 1989, pp. 59-87.

[26] Fletcher, R. and Powell, M.J.D., "A rapidly convergent descent method for minimization". Computing, Vol. 6, 1963, pp. 163-168.

Document Information

Title:	Using Simulation to Analyse the Impact of Software Requirement Volatility on Project Performance
Date:	January 25, 2000
Report:	IESE-003.00/E
Status:	Final
Distribution:	Public