Fraunhofer Gesellschaft

Fraunhofer-Institut
Experimentelles
Software Engineering

Fraunhofer-Institut
Angewandte
Informationstechnik

Fraunhofer-Institut
Techno- und
Wirtschaftsmathematik

Simulation-based Evaluation and Improvement of Software Development Processes

SEV Progress Report No. 1

Authors:
Jürgen Münch
Thomas Berlage
Thomas Hanne
Holger Neu
Stefan Nickel
Sascha von Stockum
Andreas Wirsen

Supported by the BMBF Project SEV
(Project Number 101842)

IESE-Report No. 048.02/E
Version 1.0
August 16, 2002

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

Systematic selection of appropriate processes, methods, and tools for the development of high quality software requires knowledge about their effects under varying project conditions. Up to now, the selection has essentially relied on subjective experience, empirically gained experience from previous projects, or experience from expensive controlled projects. This results in the situation that decisions concerning alternatives, in particular, are only insufficiently supported. The goal of the project SEV (Simulation-based Evaluation and Improvement of Software Development Processes) is the development of a simulation platform for software development processes and experiments. Process simulation is used to support decisions on process alternatives for a project on the basis of existing knowledge. Thereby, new development knowledge can be gained faster and more cost effectively. This progress report documents basic work towards the development of the simulation platform. In particular, application scenarios for the platform are described, an overview of an initial approach for using the platform is given, visualization requirements and concepts are described, simulation requirements and techniques are sketched, a survey of state-of-the-art simulation software is given, and an initial model for an example key software development process is sketched. Finally, a business model for applying the platform in industrial practice is described.

Keywords: software process simulation, software process improvement, controlling, decision support, software process modeling, software process planning, software process visualization, and quality assurance

Table of Contents

1	**Project Objectives**	**1**
2	**Work Performed to Date and Results**	**3**
2.1	Application Scenarios	3
2.1.1	Scenario 1: Decision Support	3
2.1.2	Scenario 2: Software Process Improvement	4
2.1.3	Scenario 3: Training Support	4
2.2	Initial Method	5
2.2.1	Systematic Elicitation of Process Knowledge	6
2.2.2	Process and Quality Modeling	7
2.3	Simulation Requirements and Techniques	8
2.3.1	General Remarks	8
2.3.2	Continuous Modeling: Software Development and Macro Simulation	9
2.3.3	Discrete Modeling: Software Development and Logistics Simulation	10
2.4	Visualization Requirements and Concepts	12
2.4.1	Requirements	12
2.4.2	User Models	13
2.5	Tool Support	15
2.5.1	Data Mining and Knowledge Builder	15
2.5.2	Simulation Tools	16
2.5.3	Optimization Tools	18
2.5.4	Other Tools	18
2.6	Initial Model for Software Inspections	19
2.6.1	Purpose of the Model	19
2.6.2	Process Model of an Inspection-based Process	19
2.6.3	Identification of Cause–Effect-Relationships	22
2.6.4	Building a Simulation Model	28
2.6.5	Potentials for Optimization	37
2.6.6	Further Plans	38
3	**Dissemination of Results**	**39**
3.1	Dissemination Activities	39
3.2	Business Model	39
3.2.1	Additional Service	40
4	**Glossary**	**41**

1 Project Objectives

The goal of the project SEV (Simulation-based Evaluation and Improvement of Software Development Processes) is the development of a simulation platform for software development processes and experiments. Simulation is used to support decisions on process alternatives for a project on the basis of existing knowledge. Thereby, new development knowledge can be gained faster and more cost effectively. The goal is divided into the following sub-goals:

1.) Forecasting and conducting 'what-if-games' for the selection of software development processes and development approaches shall be supported. The decision about process alternatives for a concrete project shall be reached on the basis of existing knowledge. Results can be used for project planning, systematic improvement of processes, and risk analyses. In addition to this, the integration of optimization algorithms promises the identification of potentials for further improvements of software development processes, which with purely empirical methods, could only be identified at high costs.
2.) The costs of new experiments shall be reduced by simulation. Well-understood parts of experiments can be simulated as well as accompanying processes such as technical processes. By analyzing results of simulation runs (e.g., by performing sensitive analyses), fields for real experiments can be identified.
3.) Practice-oriented teaching and training of project managers and planners shall be supported. Planning and project scenarios can be executed and the effects of planning decisions can be visualized graphically.
4.) The results of the project in the form of knowledge about simulation based process evaluation and improvement of software development processes shall be transferred into industrial practice. This requires a method for fast elicitation and modeling of processes as well as a business model on how to apply the knowledge in industrial contexts.

The main focus is on decision support. Better support for teaching and training is regarded as a side effect of the project.

To reach the goals of this project, a simulation platform consisting of an evaluation and improvement method, an integrated tool environment, and a simulation cockpit for the effective application of the method is going to be developed. The simulation platform should effectively support project managers and planners, people performing experiments, process engineers (in the context of process improvement), and trainers (in the context of practice-oriented teaching). Using the simulation platform promises the following advantages:

- Cost reduction by simulating software development processes and human behavior.
- Cost reduction by better selecting and focusing the scope of real experiments.
- Coupling with optimization methods promises results that could not be obtained by varying the impact factors in real or simulated experiments.
- The demonstration of the benefits of new methods in the context of an industrial development environment.
- Practice-oriented learning with respect to project planning and management can be performed in a scenario-oriented way.
- The simulation platform with the corresponding method can be integrated into process improvement programs.

Essential scientific challenges are:

- The development of a basic set of simulation models, which comprise the basic elements for the modeling of software development processes. Scientific questions address an appropriate modularization and combination techniques.
- The cost effective achievement of sufficient validity of the simulation models for the user.
- The selection of an appropriate simulation technique for different software development processes and tasks.
- The development of a method for the use of simulations for identifying process improvement potentials.
- The development of simulation techniques for the optimization of iterative processes.

The simulation of software development processes is an upcoming and innovative research field. Important issues of this project are

- the modularization of simulation models in order to allow for fast and cost effective modeling and the construction of large-scale models,
- the development of guidance that helps to select appropriate modeling techniques according to the purpose, the scope and the key result variables,
- the development of visualization techniques specialized for the simulation of software development processes,
- the identification and exploitation of potentials for optimization,
- the possibility of estimating the importance of model parameters and their sensitivity, and
- the use of simulation for the definition of real experiments.

2 Work Performed to Date and Results

2.1 Application Scenarios

In the following, typical application scenarios for the simulation platform are defined.

Each scenario definition is structured according to the following schema:

- Name: Identifier for the scenario
- User: Who uses the scenario?
- Purpose and goal: Why shall the scenario be executed?
- Benefits: What kind of benefits does the user have from the simulation?
- Steps of the execution: What steps are conducted?
- User interface (GUI): Description of user interaction and visualization
- Scientific contributions: Solutions contributing to open research topics

2.1.1 Scenario 1: Decision Support

- Name: Decision support
  As an example, the selection of an inspection technique is chosen.
- User: Project manager, project planner.
- Purpose and goal: Decision support, where in the overall development process (e.g., after design and after coding) inspections should be performed and what kind of inspections should be performed. The relation between the effort spent for inspections and their benefits is interesting.
- Benefits: Gives a better basis to decide on when and what kind of inspections should be performed and demonstrates the benefits of the inspections. Alternative inspection techniques can be compared in the project situation without cost effective real experiments.
- Steps of the execution:
  Preparation:
  1. Modeling of the project-specific development process and different inspection processes.
  2. Development of simulation models.

  Decision support:
  1. Presentation of alternatives.
  2. Selection of alternatives.
  3. Visualization of the effects in the application context (several simulation runs).

Usage:

1. Introduction of inspections based on the findings of the simulation.
2. Verification of effects using measurement programs.

- User interface (GUI):
  Presentation of important dependent variables (e.g., time, cost, and defect detection rate) with respect to the overall process
- Scientific contributions:
  - How to cost effectively achieve sufficient validity of the simulation models for the user.
  - How to model process alternatives.

2.1.2 Scenario 2: Software Process Improvement

- Name: Software process improvement
- User: SEPG (software engineering process group), responsible for quality
- Purpose and goal: Decision support concerning the impact of a process improvement measure on costs, quality, and time to market. Allows for assessing the effects of improvement measures and their prioritization (e.g., in the context of CMM improvement activities; CMM is the abbreviation of Capability Maturity Model for Software).
- Benefits: Cost effective and goal-oriented selection and prioritization of process improvement measures before implementation.
- Steps of the execution:
  1. Modeling of the project-specific development process.
  2. Identification of alternative improvement measures.
  3. Construction of simulation models of the development process.
  4. Integration of improvement measures into the models.
  5. Comparison of the simulation results.
  6. Implementation of improvement measures.
- User interface (GUI):
  Comparison of effects with respect to process and product attributes of interest (with and without the improvement measure).
- Scientific contributions:
  Development of a process for the use of simulations for the identification and assessment of improvement measures.

2.1.3 Scenario 3: Training Support

- Name: Training support for project managers and project planners
- User: Trainees for project management or project planning (persons with low experience in these tasks, students)

- Purpose and goal: Practice-oriented demonstration of the behavior of software development processes, especially the influences of planning decisions and process changes. Two situations are considered: 1.) Demonstration of the effects of typical situations in software development projects (e.g., late staffing, inspections vs. testing); 2.) Executing planning and project scenarios wherein the user can vary parameters and the effects are clearly visualized. A task could be to vary the parameters in a way that a defined behavior of the model is achieved (e.g., shorter development time at same quality).
- Benefits:
  - Fast gaining of experience without performing real projects or experiments.
  - Good identification of the importance of project management decisions.
  - Identification of side effects of decisions.
- Steps of the execution:
  1. Presentation of the situation (problem, process, context).
  2. Description of the effects under consideration.
  3. Explanation of cause-effect-relations.
  4. Simulation with parameters defined by the user.
  5. Visualization of the effects and their causes.
  6. Simulation with different parameter values.
  7. Variation of the parameters in a way that a specific behavior is achieved.
- User interface (GUI):
  Comparative graphical representation of the effects and their causes.
- Scientific contributions:
  - Visualization of the causes.
  - Supporting empirically based learning with simulations.

2.2 Initial Method

The Initial method comprises the development of the whole method consisting of integrated techniques for process elicitation, process and quality modeling, simulation modeling, combining real experiments with simulation, and goal-oriented analysis. As a first result, a framework for the method is specified.

For the framework, the following levels can be distinguished:

- Model pool level: On this level, modular process and simulation models are developed, maintained, and stored on different abstraction levels. The models on this level are not project specific.
- Project level: On this level, project specific models are built and integrated.
- Application level: On this level, simulations are performed for different purposes (e.g., decision support, training).

- Real world: On this level, results of simulation runs are used to influence real world processes (e.g., prescriptive models based on experience gained with simulation are used to guide real development processes).

For the framework, the following steps can be distinguished:

- Selection: In this step, appropriate modeling elements are identified with respect to the goal and context of the tasks to be performed.
- Instantiation: In this step, the modular models are tailored to the project context.
- Integration: In this step, the modular models are combined into an integrated simulation model.
- Operation: In this step, the simulation runs are performed.
- Application: In this step, consequences of the simulation runs are implemented in the real world process.

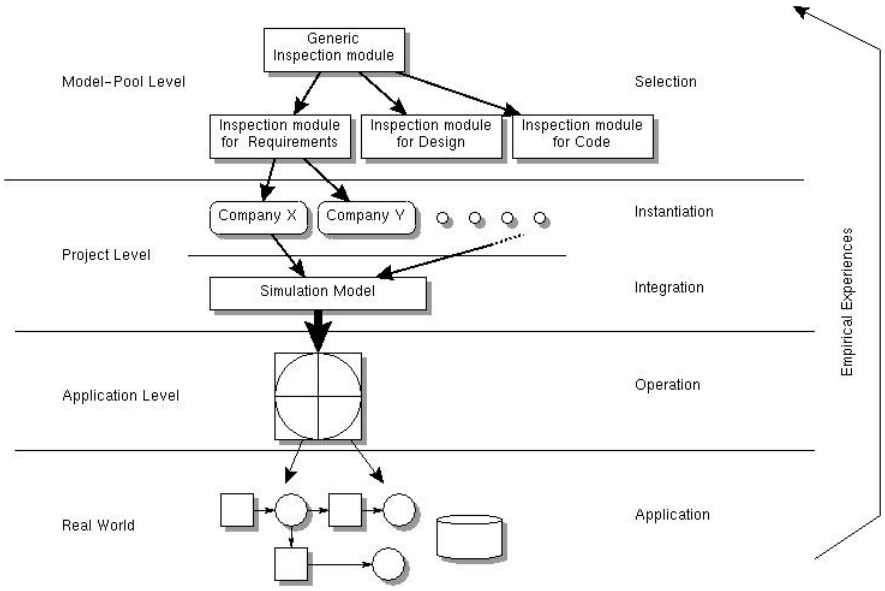


Figure 2.1 Method Framework

2.2.1 Systematic Elicitation of Process Knowledge

The Systematic Elicitation comprises the development of techniques to elicit process knowledge. Software process modeling is the task of developing software process models. This can be done either descriptively (i.e., capturing the structure and behavior of real processes) or prescriptively (i.e., reasoning about processes just from experience with the intention that the developed models guide software developers in a project).

We propose as an initial technique an elicitation approach that is concept-oriented, i.e., the elicitation of process knowledge follows the sequential gathering of information for specific process concepts. We organize the modeling as a seven-step activity that has been demonstrated to be useful for both descriptive modeling and formal modeling of standard descriptions. The steps and control flows among them are described in the following. Note that the order of steps is sequential except when specifying the control flows by attributes and criteria. Product flows are specified during steps 3 and 4. The steps are described in [VM97] and [VBG+95].

2.2.2 Process and Quality Modeling

Process and Quality Modeling comprises the identification of appropriate techniques for explicitly representing software development processes. We propose to use formal models to describe software development processes. The following advantages of formalized software engineering models (using formal process modeling languages) can be identified:

- Formal process models facilitate the creation and modification of consistent software process descriptions. The formality of the models allows computer-aided processing of the models. Process modeling environments or tools (e. g., SPEARMINT (Insert reference.)) support the creation and modification of process descriptions. The use of formal models facilitates both the consistent integration of separately modeled views to a comprehensive model, as well as the realization of automatic filters, which can select process information according to certain perspectives and present this information in an appropriate style. Since the standards can be modeled and checked from different perspectives, a higher degree of correctness and better capture of real world phenomena are achieved. This becomes more and more important when tailoring (i.e., adopting, modifying, and integrating) processes to project-specific goals and characteristics is required, leading to process variants.
- Formal process models are appropriate means for storing software development knowledge. Reusing experience (e.g., products, process models, prediction models, project plan fragments) is a key to systematic and disciplined software engineering. Although there are some successful approaches to software product reuse (e.g., class libraries), improvement should comprise the reuse of all kinds of software-related experience, especially process-related experience. Formal process models are means of capturing the relevant aspects of real world activities. They are easier to maintain than informal descriptions and they can be stored using various structures (e.g., type hierarchies, clusters of domain specific assets).
- Formal process models enable sophisticated analyses. The formal style of process description allows several kinds of (possibly automated) analyses, which can be performed before the project starts, during process execution

and in a post-mortem fashion after project termination. Before the project starts, project plans can be analyzed statically (e.g., consistency checking) or dynamically (e.g., risk analysis through simulation, or what-if analysis). Analyses during enactment comprise project tracking and trend analyses so that adjustments can be done (e.g., by providing measurement-based feedback). Post-mortem analyses (e.g., statistical tests) are used to evaluate the current practices in the project context. Weaknesses are identified and processes should be analyzed with respect to the goals of the project. Also, deviations between the actual process performance and the models should be examined.

- Formal process models are the basis for process-sensitive software engineering environments. Process-sensitive software engineering environments interpret process models in order to automate parts of the development process and to guide developers by informing them about the state of the project. Although extensive research has been undertaken in the past, only few products are available. Moreover, they have shortcomings that do not allow for successful application in larger projects. Process-sensitive software engineering environments can be seen as a future process support technology, which will provide services such as automated developer guidance, on-line coordination, communication support, and process management support.

Quality modeling can be regarded as part of process modeling. Attributes of processes, products and resources correspond to quality properties. So-called quality models capture cause-effect-relationships.

2.3 Simulation Requirements and Techniques

2.3.1 General Remarks

For modeling software development processes using advanced simulation tools it is useful to compare this domain with typical areas of application. As discussed in the simulation literature, there are basically two main approaches (simulation methods) and both of them are suitable for modeling software development processes. The first one is based on time continuous system simulation. This approach is usually called system dynamics. The other approach is time-discrete event-oriented stochastic simulation (discrete-event simulation; see e.g. [BC84]).

We will discuss both approaches in the following two subsections, comparing the software development processes with typical areas of applying the two paradigms in simulation. In Section 2.4.2. we discuss "macro modeling", which is usually used for large-scale systems as an approach suitable for the simulation of software development processes. In Section 2.4.3. we discuss logistics as an outstanding example of production processes [BHL90] analyzed by simulation

models, and consider possibilities to transfer corresponding approaches in simulation to software development processes. Also, specific requirements of software process simulation are discussed.

2.3.2 Continuous Modeling: Software Development and Macro Simulation

Complex systems are often analyzed on a macro or aggregate level because a detailed analysis is impossible: It is not possible or too expensive to access all data describing such a system. The data is afflicted with stochastic influences etc. Therefore, much of the complexity is not considered in the model. Typical examples of such systems are macro-economic systems, sectoral economic systems, and ecological or demographical systems.

Another feature of such systems is, however, more relevant: The variables describing the system on a macro level are usually interconnected and the existence of feedback loops frequently leads to an unexpected temporal behavior of the system. The beer game and the "Schweinezyklus" are famous examples of models leading to periodical changes of the system state, and this behavior is not obvious by just looking at the mathematical description of the system.

In the late 50s and the 60s, Forrester [Fo61, Fo68] has developed an approach for simulating systems which can mathematically be described by differential equations or difference equations. This approach is usually called system dynamics. He introduced a symbolic language characterizing model objects (variables) such as stocks (accumulation variables), flows (rates of change), faucets, and drains.

For applying system dynamics it is necessary to identify the most important variables describing a system, and to find out which temporal relationships they have, i.e., which variables influence which other variables and in which way. This approach was successfully used in a large number of areas.

For a system dynamics model of software development processes, typical stocks might be the workforce, the work to be done, the work finished, etc. Flow variables such as the productivity of the workforce or fatigue of the staff may influence the time required for a specific task. The application of system dynamics to software development on a quite aggregate level is, therefore, rather obvious. However, explicit representations of objects such as team members or tasks to be done are not possible in simple system dynamics models. Also, stochastic influences of the system are not considered.

2.3.3 Discrete Modeling: Software Development and Logistics Simulation

Logistics deals with activities relating to the procurement, transport, transshipment, and storage of goods. As generally understood, it is concerned particularly with the realization and control of the material flow (raw materials, interim and final products). In logistics, a simulation model represents the flow of goods through a manufacturing and commissioning plant. Typically, such models are based on discrete event simulation. Commercial software packages provide prespecified objects in logistics for simulating, e.g., conveyors (line objects), processing stations, buffers, stores, and sorters. The graphics animation of a simulation software shows moving units (MUs) being transported or waiting within a larger system constructed from single objects. Typically, various parameters of a discrete simulation model are stochastically influenced. For instance, processing times may result from random numbers according to an empirically measured distribution of working times.

Other parameters (independent variables) may be set by a control strategy and influence the performance of the simulated plant. For instance, the initial allocation of storage slots and their replenishment strategy affects the processing and waiting times of a commissioning system. Therefore, an exploration of the effects of (controllable but also uncontrollable) parameters is of high interest. This can be done by systematically performing simulation runs. But frequently, exact or heuristic optimization techniques for controlling parameters can be coupled with the simulation model (see 2.5.3.).

Software development processes are processes that deal with the generation of basically immaterial goods. The organization and sequence of the single processing steps is less formalized compared with the industrial production of material goods. The development of software is essentially a human, intellectual activity. Therefore, it is less clear in advance what the quality of a software development process is, how much time it needs, and what the costs are. Software engineering deals with the application of techniques for formalizing the software process in order to make it more manageable.

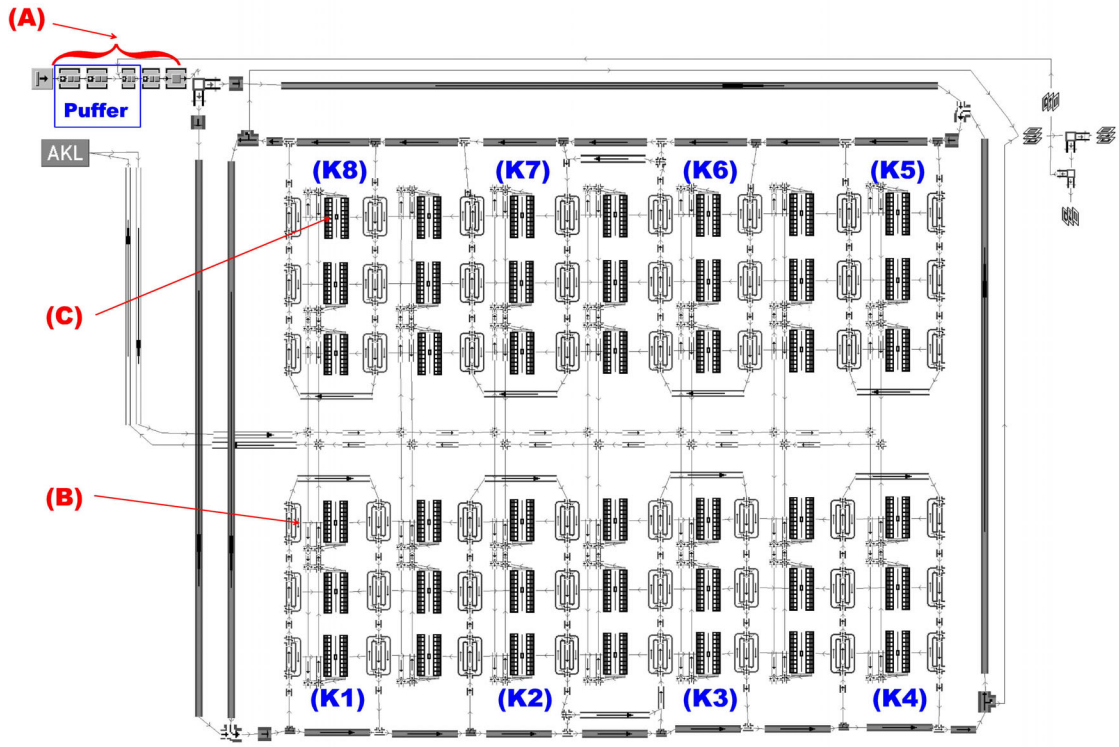


Fig. 2.2: A complex commissioning system with entrance buffers (A), main strand, and 8 commissioning loops (K1-K8) consisting of commissioning stations (working places) and racks (B) equipped with storage retrieval machines.

Typically, the process of software development is subdivided into steps such as requirements phase, design phase, code generation phase, and testing phase. In each of the phases except the testing phase some kinds of artifacts (requirement documents, design documents, source code) are generated. Possibly, there is an inspection process of an artifact for finding defects in it. Before the artifact is used as input for a subsequent phase, detected defects are reworked.

These development phases and subphases can be represented in a simulation model. In each well-defined step, tasks are assigned to staff members. The results of performing a task, i.e. quality of the artifact, required time, incurred costs, are partly stochastically influenced, partly determined by factors specific to the artifact (e.g., size), the person working on it (e.g., his/her knowledge), and general organizational factors.

Since the process of software development does not deal with physical objects to be moved or stored, concepts from logistics simulation, especially animation, can only be applied in a metaphorical sense. For instance, items (artifacts) are modeled as MUs and transferred through the various processing steps like material products through processing stations. Similarly, staff members could be

modeled as co-factors to be assigned to some items for further processing steps.

Alternatively, personnel could be represented by stationary objects. A fixed number of persons might be modeled as working stations performing jobs (with a deterministic or stochastic duration) on MUs, i.e., the artifacts to be written or inspected.

Summarizing, time-discrete event-oriented simulation facilitates detailed modeling by explicitly representing objects and focusing on the stochastic nature of processes. Compared with system dynamics modeling, these features also allow for a better understanding of complex processes.

2.4 Visualization Requirements and Concepts

2.4.1 Requirements

Work started by first obtaining requirements for the visualization and interaction demands. These depend on the particular scenarios, user intentions (training, internal communication, or decision support), and the nature of the models and simulation mechanisms employed.

We have identified the following principle requirements:

1.) The models themselves need to be understood by the users.
2.) Simulation results need to be presented comprehensively.
3.) Simulation effects need to be explored interactively.

One particular requirement in software engineering simulation is that the models employed are highly abstract. Where physical processes (e.g., for plants) are simulated, the objects and processes that are modeled are directly observable in reality and thus the model structure is rather obvious. Software engineering is an extremely complex human process. The individual actions are not easily visible, the status of the objects (software modules) manipulated can not be observed directly, and cooperative human processes evade simple modeling and control.

Therefore the first premise for visualization is the strong demand to make the models employed explicit, to make them understandable, and to enable users to map the model to their perception of the software engineering process in their organization. The presentation of the models will also influence that perception. Novices need to be trained to take a particularly effective view of the engineering process. In case of process reengineering, a clear view of the cur-

rent and future process and its implications are needed to discuss and communicate changes.

Visualizing software engineering simulations is complicated by the size of the models, which do not completely fit on a single page. The choice of model is somewhat arbitrary and multiple models covering different aspects might be used. It is a challenge to model human behavior in any non-superficial manner.

Second, comprehensive visualization of the simulation results is needed. Models may incorporate many parameters and many subobjects. It is not sufficient to individually inspect particular values. A global view of the process requires parallel visualization of a whole subset of variables.

The visualization needs to cope with mixed continuous and discrete simulations. It also needs to cope with models that consist of multiple modules or classes (e.g., human performance modeling) that are used in different subsystems. Thus, a hierarchical structure can not be assumed.

And finally, the simulation results need to be interactively explored. To understand dynamic effects or the results of a decision, an immediate feedback is most instructive. As many different variables as possible need to be interactive. Once a model is understood, it is sufficient to look at selected parameters. However, models are expected to evolve as engineering processes evolve.

One of the problems arising from this requirement is that direct integration between simulation system and simulation cockpit would be desirable. However, as the project does not intend to create a simulation engine, such an interface would be proprietary for each simulation engine that is considered. To avoid that interfacing overhead, which is not feasible in the project, a more generic interface needs to be devised.

2.4.2 User Models

To devise appropriate visualization mechanisms in the simulation cockpit, we need to identify effective views used by expert software and process engineers. Two main investigations have been carried out:

1.) The analysis of possible ways to visualize model structure comprehensively.
2.) The investigation of integral and interactive data visualization methods.

First approaches are being tested with users to identify gaps for further development.

The first observation concerning model structure is that a zooming functionality is needed. That facility needs to be organized along the individual modules the

simulation model is composed of. Modules may subdivide processes, artifacts or resources, but they may also represent aspects used in multiple contexts, such as human productivity under time constraints. A number of zooming techniques exist for hierarchical structures [Card99], but special consideration is needed for modules used repeatedly.

The second observation is that in software engineering, in addition to the image of the software process, we need to visualize the structure of the "material" (the software components, their status, and their possible reuse) and the structure of the "resources" (the development teams and their organization), as both structures and dependencies in those structures influence process performance.

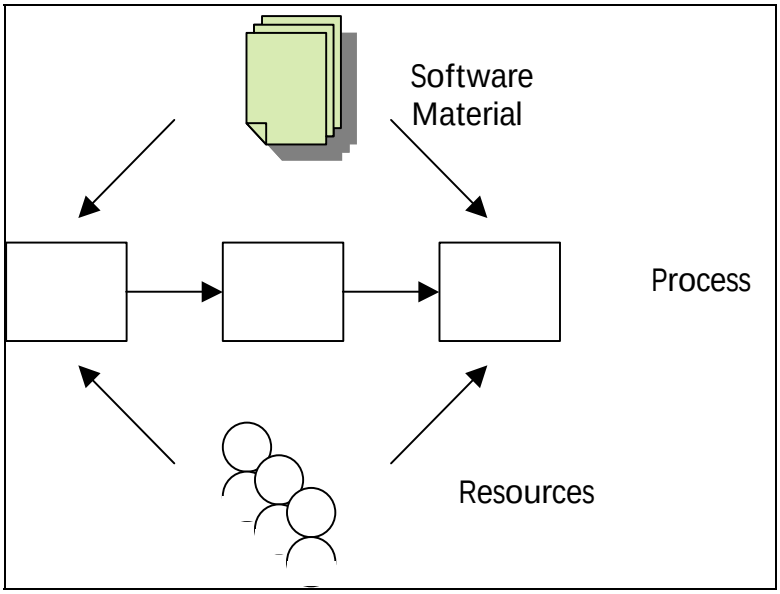


Fig. 2.2 Components that need to be visualized.

By observing and interviewing software engineering experts, effective views on a model will be identified (viewpoint analysis). For different purposes, experts usually select different aspects or perspectives. Such a selection and composition of model aspects is called a viewpoint. The viewpoints identified can then be translated into a visualization that conveys the rationale for each viewpoint (for initial understanding) and that allows quick access to the other viewpoints (for exploration).

To illustrate dynamic transitions in a model visualization, animation effects need to be incorporated to enable fluid perception. Discrete simulations need a large number of events to comprehensively evaluate model performance, while visualization needs a certain latency to perceive changes. Animation techniques can be employed to improve visualization of dynamic behavior. [Gonz96]

(2) To effectively visualize simulation results, in particular the results of continuous simulations, an aggregation mechanism is needed. Aggregation avoids tedious inspection of individual simulation items. Data can be exported from commercial simulation packages as database files. The InfoZoom system [Spen96, see 2.5.4] lets the user explore such files interactively and enables the user to zoom in on particular aspects. In this way, batch simulations (varying multiple input parameters) can be presented.

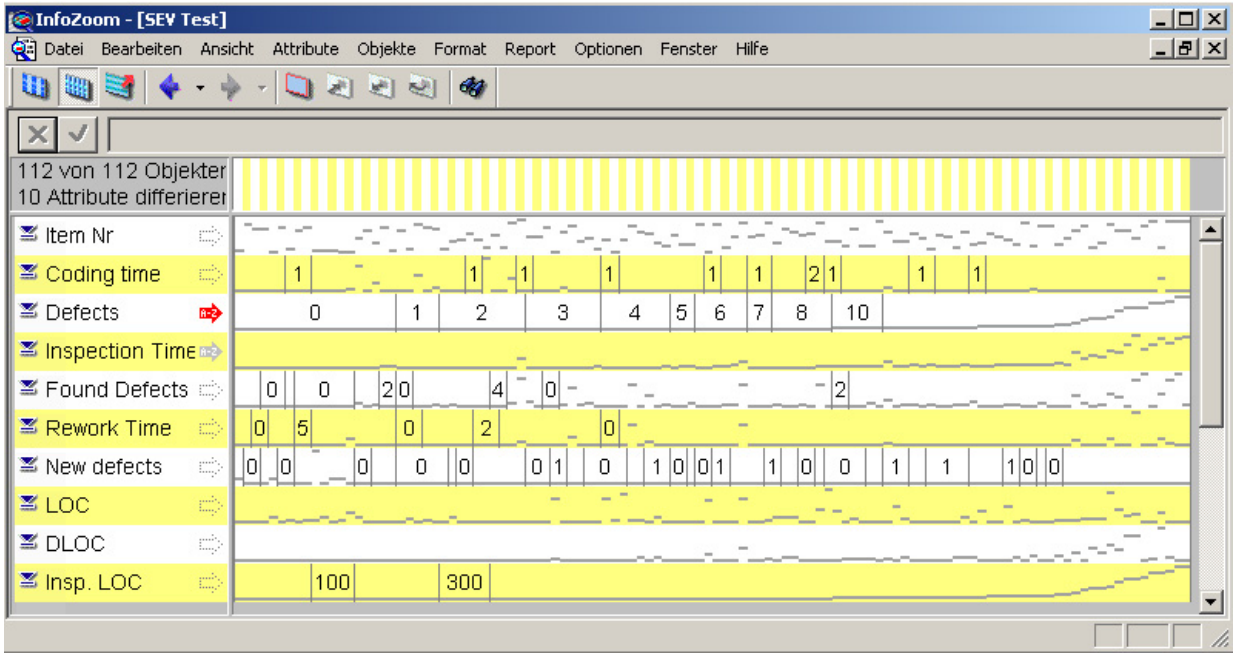


Fig 2.3 InfoZoom compressed presentation of simulation results.

2.5 Tool Support

Results on ascertaining tools for data analysis, simulation, and optimization to be used within the project are presented in the following.

2.5.1 Data Mining and Knowledge Builder

As already mentioned, software development is a process that is subdivided into different phases such as 1) requirement phase, 2) design phase,3) code generation phase, and 4) testing phase. In each of the phases 1) –3) a document is generated, inspected, reworked before it enters the next phase. A lot of factors like experience or fatigue of the developers have an influence on the time needed for the phase and the resulting quality of the document. It is not always clear how the different factors that determine the duration and overall

quality are interacting. Therefore, rules have to be developed using historical data and the knowledge of software engineers. In other fields of applications, data mining tools and knowledge builders are being successfully used. An introduction to data mining can be found for example, in [Lusti99] or [Nauck96].

2.5.1.1 Data Mining

Data mining tools are used to discover patterns in historical data. There are different types of data mining tools available, which can be separated in three technologies [Attar].

1.) Query and reporting tools
Applying these tools, it is possible to find answers to queries already being suspected. In software engineering, "cause effect diagrams" are an effective way to describe relationships between the different factors. These could be used for the investigation of patterns.
2.) OLAP tools
In addition to the previous tools, these tools allow to interrogate multidimensional databases speedily and graphically.
3.) Data mining tools for pattern recognition
These tools automate the process of discovering pattern in data. They enable business goal driven discovery, which means that, in addition to the previous tools, the user can ask for patterns relating to low costs, for example.

The tool used in the project is XpertRule Miner from Attar software, which supports all the features described above. It is possible to import data from Microsoft Excel or other OBDC compliant data sources. The tool provides a graphical environment for supporting all the stages of the data mining process.

2.5.1.2 Knowledge Builder

In addition to the data mining tool, XpertRule KBS and XpertRule Knowledge Builder will be used to capture the knowledge of the software developers and software engineers to create rules, which are then integrated into the design process.

XpertRule consists of a graphical development environment that makes it easy to prototype, build, maintain, and test knowledge based systems.

2.5.2 Simulation Tools

Based on two main sources, a survey of state-of-the-art simulation software has been performed: An extensive survey on software packages for simulation has

appeared in a recent issue of OR Today [Sw01]. This is a continuation of a former survey [Sw99] of tools for discrete event systems simulation and related products. Furthermore, in 1999 a survey study was performed at ITWM. This study includes a detailed comparison of features of the packages eM-Plant, ARENA, Taylor II, and Taylor ED, which are suitable for logistics simulation. Based on these market surveys, two main results are: 1) Basically, all simulation tools considered in these surveys provide interesting features, at least for discrete-event simulation. 2) Some of the packages can be excluded by being too specialized and too expensive. For these reasons, we considered the literature on software process simulation according to the selection of suitable packages.

The system dynamics approach is mostly used for "macro-modeling" and supports feedback loops. The second approach is usually better suited for more complex and detailed models. Graphical animation is supported as well as stochastic influences on model data.

Most software process simulation models discussed in the literature are based on the system dynamics approach. Examples are given in [LAS97], [MaTa00], [MeCo97], [PfLe00], or [PKR00]. Different software tools are available for system dynamics models. Quite often, the commercial software package VENSIM is used, e.g., in [PfLe99] and [PfLe00]. This tool includes, among other features, a visual modeling interface and an optimization component. We have tested VENSIM by applying it to some initial modeling of a continuous software process.

The commercial simulation software package EXTEND provides the possibilities of both continuous and discrete modeling. Therefore, it allows for more detailed and stochastic models than a pure system dynamics type software. The EXTEND software is quite frequently used for software process simulation: A complex continuous EXTEND model has been worked out in [Rus97]. Due to the need of using discrete modeling features in a mainly continuous model, EXTEND has been used in [RCL99]. A hybrid (continuous and discrete) model is developed in [MaRa00]. In [ChSt00], for instance, a simple discrete model of a software requirements development process is discussed.

Other simulation packages used in the literature for software process simulation are, for instance, SIMULOG/QNAP2 [Dola00] (for a hybrid simulation model), SESAM, or ITHINK (a system dynamics software).

Because of the frequent usage in software process modeling and the provision of features for discrete and continuous modeling, we decided to focus on the application of EXTEND for the project. The well-established tool VENSIM is used in the project in a supplemental way.

Besides that, the optimization department of ITWM has been using the eM-plant simulation software (and its predecessor Simple++) for various industrial

projects for a couple of years. The utilization of this package, for which several licenses are available, is considered as well.

2.5.3 Optimization Tools

Both simulation packages chosen for the project, EXTEND and VENSIM, include optimization components that might be useful for tuning a simulation model (see 2.6.3.). The most common approach at ITWM for combining simulation and optimization is, however, as follows: If possible, sub-problems are solved prior to the simulation phase using a traditional exact solution method usually based on commercial software like CPLEX or XPRESS. If such a priori optimization is not possible, online strategies are applied during optimization. Most often these are heuristic approaches, since an exact solution would consume too much computational time.

2.5.4 Visualization Tools

Most simulation packages include visualization tools. Recent research has shown that *focus+context* techniques improve the visualization of complex relationships [Card99]. Furthermore, *dynamic queries* [Card99] is a mechanism to explore a multidimensional space, for example, of simulation results.

InfoZoom [Spen96] is a tool that combines both techniques to visualize and explore complex tables. The user can quickly focus in on arbitrary subsets of the data and thus get an interactive understanding. Similar to a spreadsheet, InfoZoom also includes the capability to generate diagrams.

In the project, InfoZoom will be used to visualize and explore simulation results exported from a simulation package. A direct connection using ODBC will also be investigated.

2.5.5 Other Tools

As a general purpose tool providing unified modeling language (UML), the "Rational Rose Data Modeler" is used for supporting the collaborative development of documents within the project.

In addition to the tools of Attar Software, ITWM uses MATHWORKS Matlab in combination with the neural network, system identification, fuzzy and simulink toolboxes to develop pattern recognition algorithms, fuzzy systems and simulation tools (see for example [Weiß99]). Since 2000, the ITWM department Adaptive Systems has been an official partner of MATHWORKS, Inc.

2.6 Initial Model for Software Inspections

2.6.1 Purpose of the Model

This section sketches an initial model for an example key software development process, i.e., a software inspection process. The goal of the initial model is to understand the relationships between effort, duration, the number of defects detected, and their impact factors.

2.6.2 Process Model of an Inspection-based Process

In the following, a process model for code inspections is described. A detailed description of this inspection process can be found in [EbS93]. The description is based on a product-focused V-Model. Figure 2.2 shows a part of the overall development process, in particular, the design and implementation of a component. This part needs to be instantiated for each component of the product.

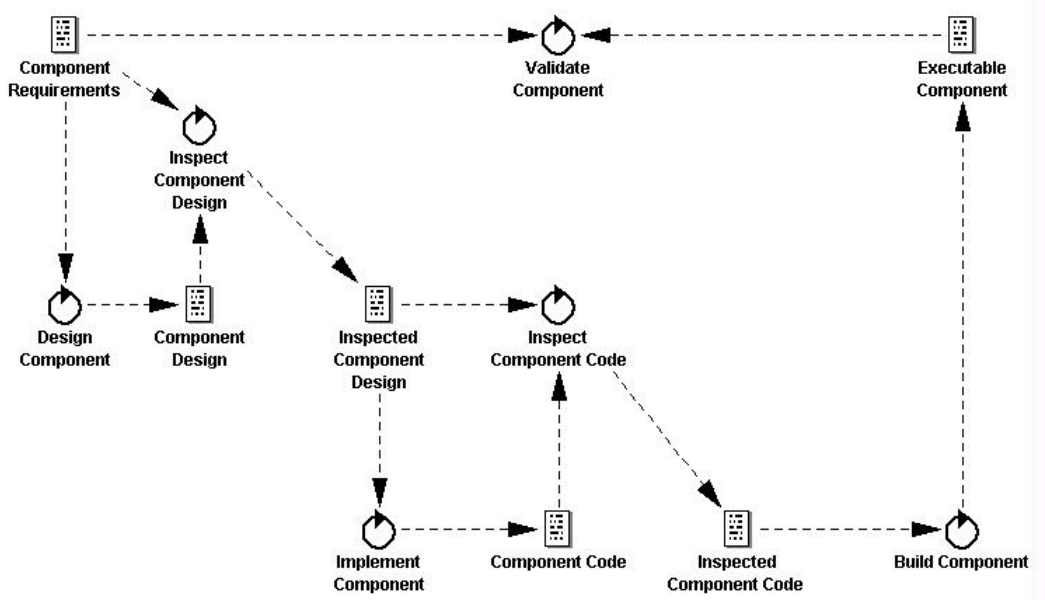


Figure 2.2 Design and implementation of a software component

The inspection of the component design can start if the creation of the component design document is completed. Additionally, the component requirements are needed as inputs for this inspection process. Output of the component design inspection process is an inspected component design, which is a version of the component design document that is corrected with respect to the defects found during the inspection process. The inspection process of the component code is analogous.

Figure 2.2 shows the product flow. We differentiate between the documents (component design and component code) before and after the inspection to emphasize the fact that the document status has changed. This excerpt of the development process needs to be instantiated for each component of the system to be developed. The initial simulation model focuses on the activity 'inspect component code'. This activity is refined, meaning it is described in greater detail in another product flow. Figure 2.3 shows the process of a single inspection performed for one code component. In the following, we describe the activities and roles in the inspection process.

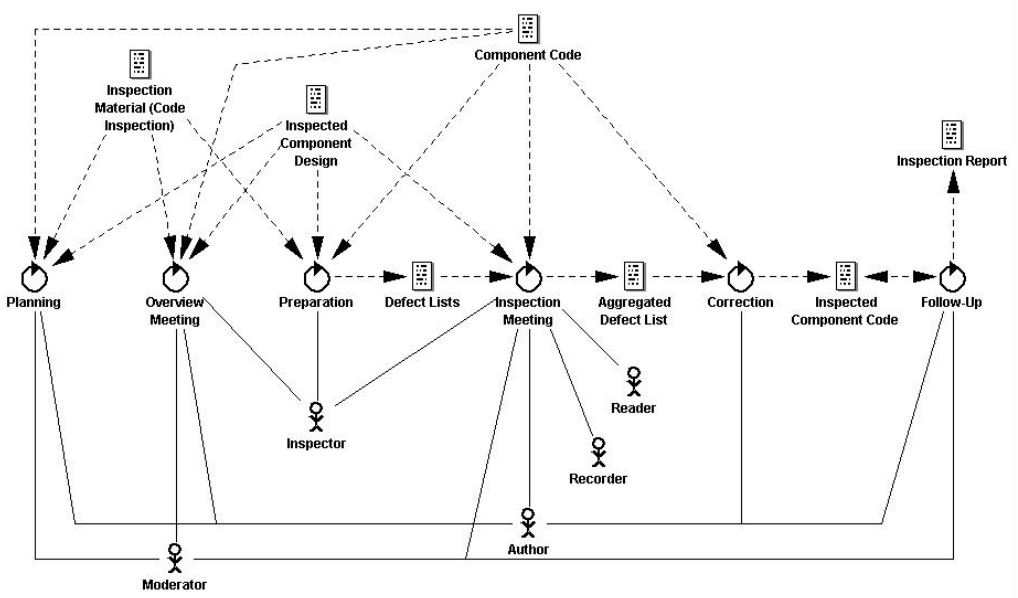


Figure 2.3 Refinement of the component code inspection activity

2.6.2.1 Inspection stages

The goal of the inspection process is to eliminate defects from a work product. For this, the following activities are performed.

1.) **Planning:** The main objective for the planning activity is to organize the subsequent steps of the inspection process. After finishing a work product, the author collects all necessary material for the inspection and selects a moderator for the inspection. Together they determine the other inspection participants and the moderator schedules the following activities with respect to time and staffing. He or she also distributes the inspection materials (e.g., the component code, the inspected component design and other inspection material like checklists).
2.) **Overview meeting:** If an overview meeting is held, the objective is the education of the inspectors (e.g., to explain algorithms or complex rela-

tions). The author presents the material to be inspected to the other participants. The moderator arranges and moderates the overview session.

3.) **Preparation:** The preparation is an individual activity that is performed separately by each participant. The participants become thoroughly familiar with the material and note any suspected defects they can identify. This activity is often also referred to as ‘detection’.
4.) **Inspection meeting:** All roles participate in the inspection meeting. The moderator conducts the meeting and the roles of the reader and the recorder are performed by members of the inspection team. The reader paraphrases the product and every inspector mentions the potential defects he/she found. The recorder records all defects that are accepted as defects in a defect list. This activity is often also referred to as ‘collection’.
5.) **Correction:** The objective is to remove all defects found in the artifact. The author performs the correction.
6.) **Follow-up:** The objective of the follow-up is the verification of the rework that was done. The moderator certifies that the corrections have been completed and creates an inspection report.

2.6.2.2 The Different Roles in an Inspection Team

An inspection team is formed by several persons who perform the following five roles. Together they inspect a work product to identify the defects in the work product.

1.) Author: The person produced or is responsible for the changes in the product to be inspected. The author can not fill the roles of moderator, reader, or recorder in the inspection team.
2.) Moderator: The moderator ensures that the inspection process is followed and the members of the inspection team perform their tasks in the inspection process.
3.) Reader: The reader is a member of the inspection team who guides the inspectors through the work product during the inspection meeting. This role can not be performed by the author.
4.) Recorder: The recorder writes the defect list during the inspection meeting. He adds only the defects that are identified as defects by the inspection team and classifies the defects with the moderator.
5.) Inspector: All members of the inspection team can be inspectors, including the author. The inspectors should only identify defects and should not search for solutions.

A more detailed description of the roles in a software inspection process can be found in [EbS93].

2.6.3 Identification of Cause–Effect-Relationships

Before we can build the simulation model, the influencing factors have to be identified. Software development is a human based and highly dynamic process and, therefore, difficult to analyze and to simulate. We will apply parts of the System Dynamics technique (see Section 2.3.2). Causal diagrams are used for analyzing the relationships of the attributes in the inspection process.

At first, we will describe the influencing attributes of an inspection process.

In [LaD89] the authors differentiate the core inspection concepts and relationships into five dimensions. This description is used to identify potential impact factors on the variables of interest (e.g., effort, duration, and number of detected defects). These five dimensions are the

- technical dimension, which covers the different methodological variations;
- managerial dimension, which characterizes the relationships between an inspection and project issues;
- organizational dimension, which characterizes the relationships between an inspection and an organization;
- assessment dimension, which describes how to evaluate an inspection;
- tool dimension, which deals with tool support for inspections.

For the initial simulation model we concentrate on the first three dimensions. The fourth dimension describes assessment issues. We exclude the tool dimension for the initial model because [LaD89] describe that the use of tools is limited.

2.6.3.1 Impact Factors

The technical dimension covers the different methodological variations. The following impact factors have been identified:

- Process: The process describes the different variations and similarities among the methods with respect to the selected reference process.
- Product: It is differentiated which type of product is to be inspected, e.g., a code document or a requirements document. Also the maturity, the size, and the complexity of a product are important attributes. These attributes are the product characteristics.
- Reading technique: The reading technique describes the technique that is used during the preparation phase of the inspection. We concentrate on reading techniques, because reading is one of the key activities for individual defect detection [Bas97]. Examples for reading techniques are ad-hoc reading or checklist based reading.

- Team size: The team size interrelates with the process, because variations of the process influence role definitions and workforce for the inspection.

The organizational dimension characterizes the relationships between an inspection and an organization. If necessary, these influences have to be modeled but this requires individual adaptation of the model according to the situation found in the organization. Especially the human factors and connected factors have to be addressed individually. Some can be clustered as characteristics of the team. Besides the number of people and the number of teams involved, the number of inspectors and their experience are important impact factors.

- Experience of inspectors: The experience of the inspectors is an obvious impact factor. It can be divided into three kinds of experience: inspection experience, development experience, and domain experience.
- Number of inspectors: There is a relationship between the number of inspectors and the number of defects detected.
- Number of people in the inspection team: The number of all people involved in the inspection process. This also correlates with the team size.
- Number of teams: Usually only one inspection team performs the inspection. More than one team increases the duration and the effort of the inspection but also improves the number of defects detected.

In the following we will use only the experience of the inspectors and the number of inspectors, because the number of teams only fits in special cases (multiple teams inspect the same document in parallel). The number of people and the team size can be seen as the same impact factor.

In software development projects, time pressure is almost always a factor that influences people. The productivity of the team or that of an individual also has an influence on the performance of the inspection. For that reason, we define new attributes that apply to the team or to each individual person.

- Time pressure: Time pressure is a cause for overtime and will eventually result in more fatigue.
- Fatigue: A high workload and overtime for a longer time period result in fatigue effects. The inspector does not find as many defects as if he/she were rested.
- Productivity: Productivity can be divided into three different kinds of productivity: the development productivity (for writing the code), inspection or reading productivity, and rework or correction productivity.

2.6.3.2 Factors of Interest

The managerial dimension characterizes the relationships between an inspection and project issues. These attributes are often the dependant variables.

- Effort: Effort addresses the number of hours spent on an inspection. Costs are closely related because an inspection is a human-based activity.
- Duration: There are two kinds of duration, the duration of a single inspection (including planning, overview meeting, ...) and duration with regard to all inspections performed in one project. Both are measured in calendar time.
- Number of defects detected: A very important factor that can be improved by inspections is the number of defects in a product. Therefore the number of defects detected is seen as a factor of interest.

2.6.3.3 Cause Effect Diagram

A cause effect diagram displays all factors of a mental model and their relations. "In System dynamics, the term 'mental model' includes our believes about the network for causes and effects that describe how a system operates, along with the boundary of the model and the time horizon we consider relevant." [Ste00]

In Table 2-1 the link polarity in a cause effect diagram is defined. The links describe the mental structure of the model, but they do not describe the exact behavior of the variables in the model.

Symbol	Interpretation	Mathematical Definition
	All else equal*, if X increases (decreases), then Y increases (decreases) above (below) what it would have been. In the case of accumulations, X adds to Y	$\partial Y / \partial X > 0$ in the case of accumulation $Y = \int_{t_0}^{t} (X + \ldots) ds + Y_{t_0}$
	All else equal*, if X increases (decreases), then Y decreases (increases) below (above) what it would have been. In the case of accumulations, X subtracts to Y	$\partial Y / \partial X < 0$ in the case of accumulation $Y = \int_{t_0}^{t} (-X + \ldots) ds + Y_{t_0}$

Table 2-1 Link polarity: definition [Ste00]
* All else equal means that if all other variables are kept equal, then X and Y show this behavior.

In [LaD89] the authors present causal models to explain the influencing factors on inspection quality, effort and duration. The models are similar to cause effect diagrams but do not include feedback cycles. In Figure 2.4 the three diagrams are incorporated into one diagram. The three variables effort, duration of inspection, and number of defects detected in an inspection are the dependent variables. Two groups of independent variables can be identified: product characteristics and team characteristics.

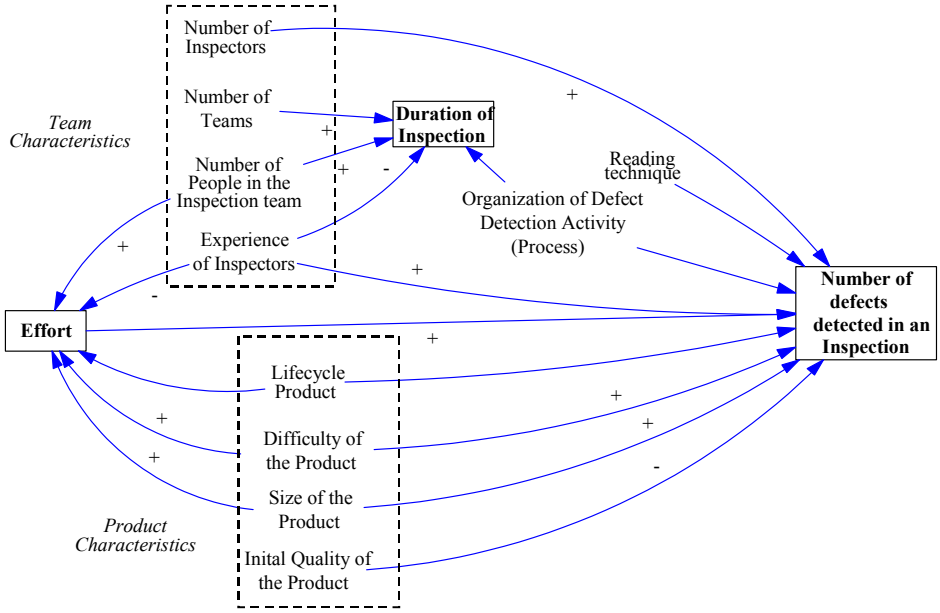


Figure 2.4 Merged Diagrams from [LaD89]

For the simulation model, some variables will be merged and new variables will be introduced to show further dynamic effects such as time pressure. Other variables effect only the model architecture, such as the organization of the defect detection activities.

Figure 2.5 shows the cause effect diagram for the code inspection process. The diagram originates from expert interviews and literature (in particular [Lad98]). The diagram introduces some new variables that form the dynamics of a simulation model. These variables will be explained in the following paragraphs.

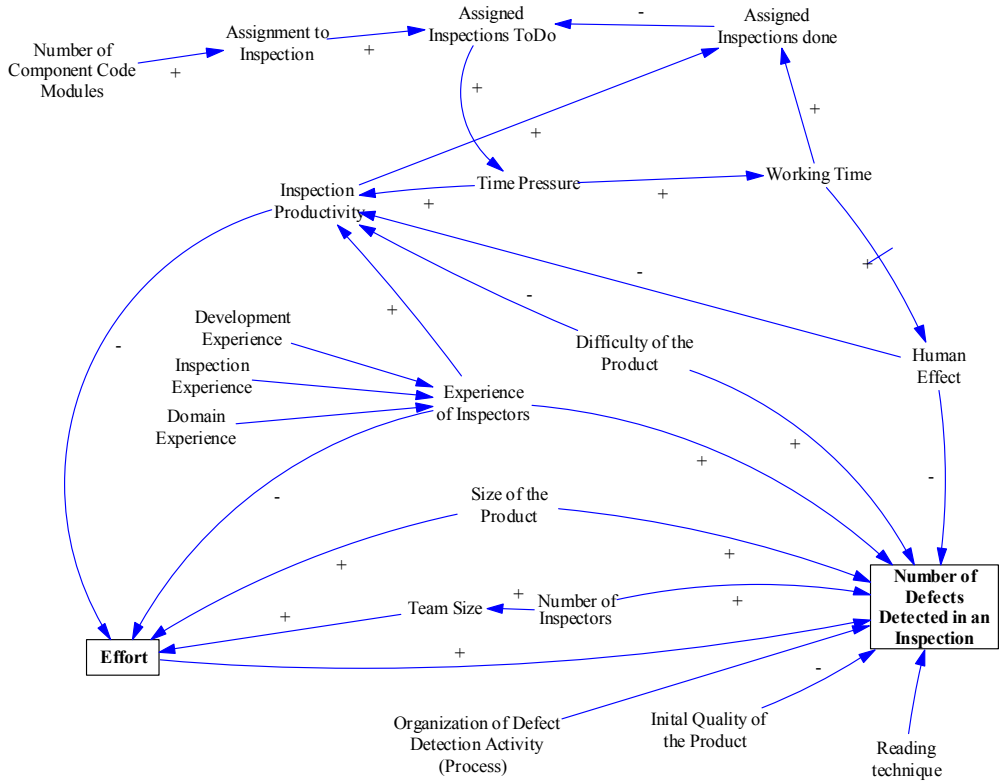


Figure 2.5 Cause effect diagram of an inspection process

Some of the variables from Figure 2.4 are missing. In the cause effect diagram the product type or life cycle product is excluded as impact factor because this cause effect diagram is for the code inspection process and, therefore, the type is constantly code. Also, the ‘number of teams’ and the ‘number of people in the inspection team’ are left out because the ‘number of teams’ applies only if two or more teams inspect an artifact in parallel and the number of people correlates with the team size. Another important factor, the duration of the inspection, is left out because time is a simulation variable. If the time for a single inspection or all inspections is needed, it can be determined from the simulation time. In the following, the variables are explained in more detail.

- Effort: Effort is the overall time spent on the inspection (measured in person days or hours). This information is also a good figure for the costs, because effort is the main cost driver for inspections. With the number of defects found the costs per defect can be computed.
  Measure: Person days (PD) or person hours (PH)
- Experience: The participants of an inspection have a different level of experience. The type of experience is also different, e.g., development experience, inspection experience, and domain experience. Generally, the more experi-

ence a participant or the average team member has, the more defects will be found and corrected. For measuring experience, we can introduce a measure that defines different levels of experience.
Measure: Experience level

- Difficulty of the product: Often this attribute of a product is called product complexity. The complexity of a product also influences the number of defects found. On the one hand, a complex product often has more defects that can be found. On the other hand, if not enough effort is spent to understand the product, less defects are found than possible.
  Measure: Complexity (Possibly insert some explanations and references on how to measure complexity.)
- Initial quality of the product: If a mature product is inspected that was merely changed, defect density is lower than in a newly created product and less defects will be found. Often this product attribute is called product maturity.
  Measure: has to be defined.
- Size of the product: There is a correlation between the size of a product and the number of defects it contains.
  Measure: LOC or Function Points.
- Reading technique: Depending on the reading technique, more or less defects can be found. In industrial practice three techniques are mainly used, Ad hoc, Checklist Based Reading (CBR), and Perspective Based Reading (PBR). In [LEH99] the authors compare CBR versus PBR. Generally, about 1/3 or 1/2 of the defects in a product are found.
  Measure: Integer value, each represents a reading technique.
- Team size: The number of defects found depends on the number of participants (This is a nonlinear dependency). [EbS93] recommend seven participants as the maximum team size and three as the minimum. The team size is closely related with the number of inspectors because all inspectors are members of the team.
  Measure: Number of team members.
- Number of inspectors: The number of team members who read the code in the role of an inspector.
  Measure: Number of inspectors.
- Organization of defect detection activity (process): The process has an impact that can not be clearly defined. Depending on changes a model could be changed completely or only a parameter will change.
  Measure: None
- Number of defects detected in an inspection: This is usually the main factor of interest. If rework is modeled, the number of defects detected influences the rework time for a code component.
  Measure: Number of defects detected
- Time pressure: Time pressure in a project can lead to overtime for the project members to accomplish the work, so the working time per day will increase.

Often, time pressure leads to a go / no go decision for the inspection.
Measure: to be defined.

- Working time: The working time are the hours per day that are spent working on inspections. Here a sophisticated model can simulate the behavior of a person. We simply assume 8 hours per day as normal working time. More hours will be counted as overtime.
  Measure: Hours per day
- Fatigue: If the participants endure more working hours per day for a longer period of time, fatigue will lessen the usefulness of the inspection. Fatigue is a factor that decreases the number of defects found and the inspection productivity.
  Measure: To be defined.
- Inspection productivity: Productivity can be split into three productivity types: inspection productivity, development productivity, and rework productivity. Productivity can be set for an individual or an average productivity for a group.
  Measure: LOC/hour or Function points/day
- Number of component code modules: The number of modules to be inspected.
  Measure: Number of modules
- Assignment to inspection: Not all components need to be inspected, here we can specify the percentage of the components to be inspected.
  Measure: % or an absolute number if known
- Assigned Inspections To-Do: This is the number of the inspections that really has to be done.
  Measure: Number of modules
- Assigned inspections done: These are the inspections done. This includes the rework in the following model.
  Measure: Number of modules

2.6.4 Building a Simulation Model

Before developing a simulation model, a decision has to be made on whether a continuous model or a discrete model should be built. At first we started with a continuous model and with the system dynamics (SD) tool Vensim. In the SD approach all items are scalar values. This assumption works very well for high level models. If more details are necessary, then the differences between the items have to be considered. For example, design and code items differ in structural properties like size, complexity, and modularity. Defects can also have different types and consequences depending on the phase when they are found. In accordance with this requirement the simulation tool EXTEND can also handle discrete event models besides the continuous models. Therefore, we used EXTEND for the later models.

2.6.4.1 Abstract Model

The description of the inspection process in Section 2.6.1 illustrates the structure and the different steps of the inspection process. The cause effect diagrams of the software inspection show the expected relationships between the different variables (explanatory and dependent variables). In software engineering, rules are usually developed to improve the overall software development process of a company following a Goal Question Metric (GQM) approach.
In a simulation model, additional feedback loops should be considered, such as, for example, the growth in experience of the developer during the different phases of the process. The experience influences the productivity of producing a document and the number of defects, especially those generated by the developer in the coding process, or the time needed for reading a document during the inspection process.
The relationships of all variables have to be determined in order to model and simulate the feedback in the software development process. Therefore, data mining tools or the Knowledge Builder will be used in a next step to generate the missing rules depending on the data or description of the process available.

Cause effect diagrams of all processes (coding, inspection, rework) of the considered phase are generated and it becomes quite obvious which of the variables of the inspection process also play an important role in other processes in the considered phase. It seems that especially the feedback relationships in the software development process have not been considered in very much detail up to now. Additionally, one has to consider that not all variables will be changed at the same time step. It is also possible that a team member has to interrupt his or her work in the coding process to participate in an inspection process. The extra time needed to start programming again could also be modeled depending on the experience of the programmer. Modeling such interrupts can be seen as fine tuning of the simulation model, which is not supported in many existing simulation tools and which will not be considered in an initial simulation model.

Every variable usually denotes an attribute of an object. In logistic simulation, the objects influenced in different processes were modeled by moving units (MU), where every MU has different values for the different variables. The variables of a moving unit are changed in a working station denoted here as process (coding, inspection, rework).

It seems quite reasonable to split up the variables into different groups. A quite natural splitting is:

1.) Global variables
2.) Item (document) variables
3.) Staff variables
4.) Software inspection variables

2.6.4.1.1 Global Variables

These variables are defined by the manager and can be seen as the input of the whole software development process. The variables determine, for example, the time pressure and the complexity of the different items. In the simulation software these variables should be also modeled as inputs.

1.) Product complexity
2.) Product size (number of items)
3.) Time schedule
4.) Team members (number, qualification)

2.6.4.1.2 Item Variables

A piece of code that has to be designed, coded, inspected, and reworked during a phase is called Item. All variables that appear in all working units are collected in this group. They can be handled as MUs (see Section 2.3.3).

1.) Complexity (input–output functions, database functions,…) measured, e.g., in function points
2.) Total size (lines of code)
3.) Size of new code
4.) Size of inspection code
5.) Maturity
6.) Defects

2.6.4.1.3 Staff Variables

This group simply includes variables one would associate with humans. The values of the variables are also changed in the different processes (coding, inspection, rework). Staff members can be modeled as moving units, too. In contrast to ITEM variables, these MU pass the different processes several times and are therefore feedback components of the considered phase. Additionally, they can play different roles in an inspection process (author, moderator, reader,..).

1.) Fatigue
2.) Experience in
    a.) programming
    b.) programming language
    c.) inspection
3.) Personal Productivity
4.) Personal error generation factor

2.6.4.1.4 Inspection Specific Variables

Since one goal of building up the simulation tool is decision support, the simulation tool should allow the use of different inspection techniques. Here only those variables are collected that appear in the inspection process and that might differ in the various reading techniques. Therefore, in a discrete simulation approach the variables can be seen as attributes of a working station, which can be changed interactively by the software manager.

1.) Team size (inspection)
2.) Roles (inspection)
3.) Preparation Time (inspection)
4.) Meetings (yes/no) (inspection)

It now becomes clear that a discrete simulation approach should be used for the simulation, since the state of the variables for all staff members and for all items differs. Additionally, the number of persons involved in an inspection process also varies during the process. Staff not involved in the inspection process can proceed with other jobs.

In the next step the relationships between the different (MU) objects at the working stations, which are the different processes, have to be determined. Therefore, known relationships have to be plugged into the simulation model and new rules have to be generated to describe the change of the values of the different variables in a working station, which have not been considered up to now. The choice of methods to be used for rule generation depends on the already existing knowledge or historical data available for the different phases. Furthermore, one has to mention that every time a new variable is introduced in the inspection process, one should analyze its relationships to the other variables.

To the see the drawbacks of a system dynamics approach with respect to the goal of simulating a software inspection process, we build an initial time continuous model. The main disadvantage is that one can not really differentiate between the properties of the different staff members. The only possibility in a continuous time model would be to use stochastic disturbances of the different variables. Most system dynamics simulation tools can not handle such stochastic elements.

Based on the description above we started designing an initial discrete-event simulation model for software development, which will be explained in the next subsection. Let us note that various relationships between the different variables still have to be determined. Therefore, the simulation model should be interpreted as a flow diagram for the different MUs in one phase in the software development process, which has to be refined. On the other hand, starting to build this model leads to a better understanding of the relationships of the different subsystems (working stations).

2.6.4.2 System Dynamics Model

For building a model, more information is necessary than given by the attributes of the software inspection described before. There are *entities, factors, variables, interactions,* and *operations* to be identified that describe the software development process [RCL98]. The *entities* are mainly people involved in the process, but can also involve facilities, equipment, and tools if these are crucial for the process. *Factors* have a relevance for the general software process like application domain, size, schedule, and other information that characterizes the process. The *variables* are the factors a manager could change, like the number of developers and the skill levels of those individuals. *Interactions* and *operations* increase the complexity of the model. Operations are the tasks that need to be performed in order to transfer requirements to a design or a design to code, for example. Typical operations are requirements analysis, architecture development and so on. The most challenging task is to represent the interactions among the factors and variables in a software process that are important for the model. For example: How does the exhaustion of people affect their productivity or the quality of their work? Or in the case of inspections: How does the reading technique affect the number of defects found during the detection phase of an inspection?

For our first models we used the continuous SD technique. Below the task flow of a software process is shown. This model represents one stage in the development process (e.g., coding) with an inspection.

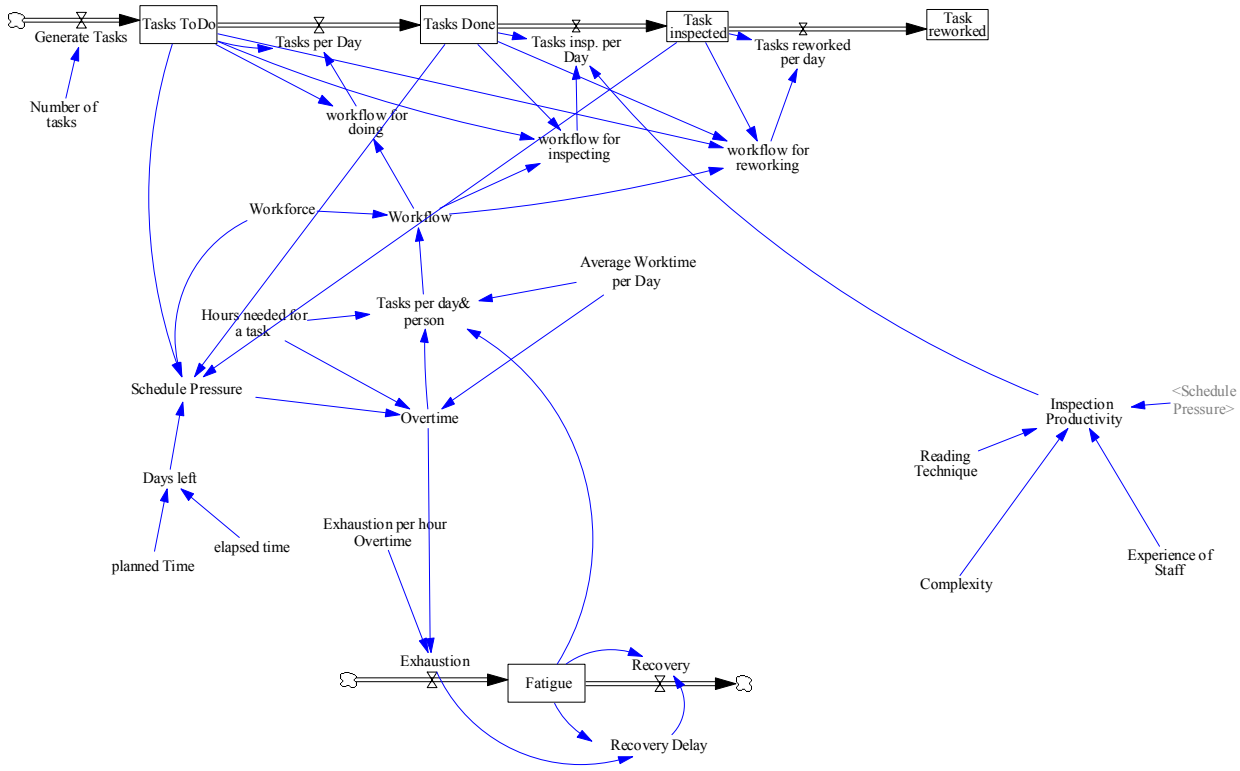


Figure 2.6 System Dynamics model of the Task Flow

As shown in Figure 2.6, only one value for the complexity of the complete product or for the number of participants (workforce) can be set. On the other hand, it is easier to model feedback loops. A typical feedback loop is the fatigue of the participants. In Figure 2.7 the diagram for exhaustion, fatigue, and recovery of the model in Figure 2.6 is shown.
There are no differences between the tasks that have to be accomplished. If more details are required, a discrete model has to be used.

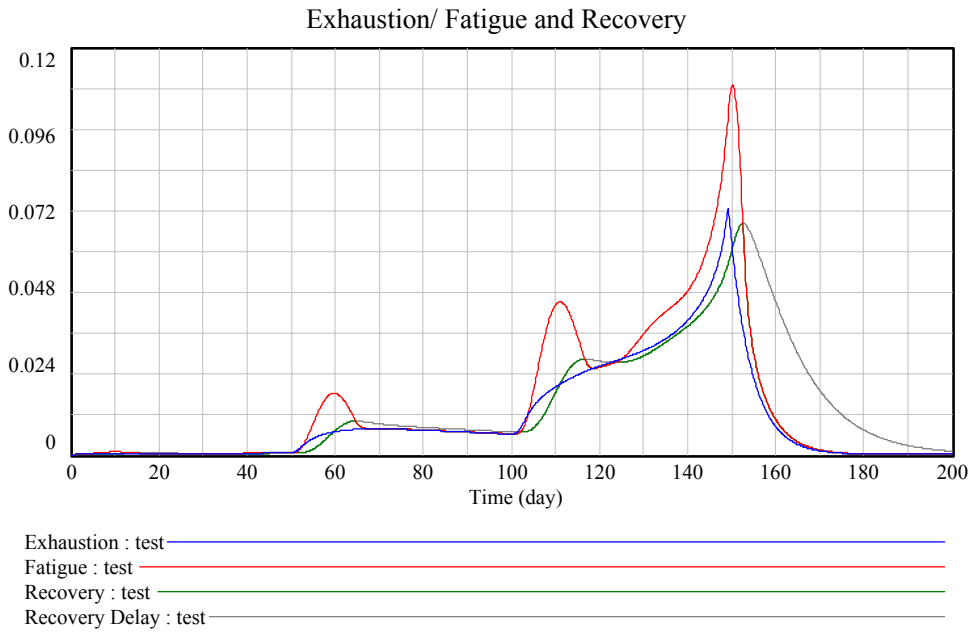


Figure 2.7 Exhaustion, Fatigue and Recovery of all participants

2.6.4.3 A Discrete-Event Simulation Model for Software Development

For developing a discrete model, at first we have to identify the entities, factors, and variables. After that the operations and interactions are described with the architecture of the model. A cause effect diagram is also useful here to capture the mental model.

As discussed in Section 2.3.3, items and staff members are modeled as moving units (MUs).These MUs have attributes that contain the dedicated variables and factors of the single items. An MU can match a group of variables of the abstract model.

A person object-represents one person with their skills and personal attributes, i.e., the staff variables of the abstract model. An item object represents a part of the work that has to be accomplished by the people. This could be a software module that has to be implemented or a set of requirements that have to be transformed into an architecture design. This type of object contains the item variables of the abstract model.

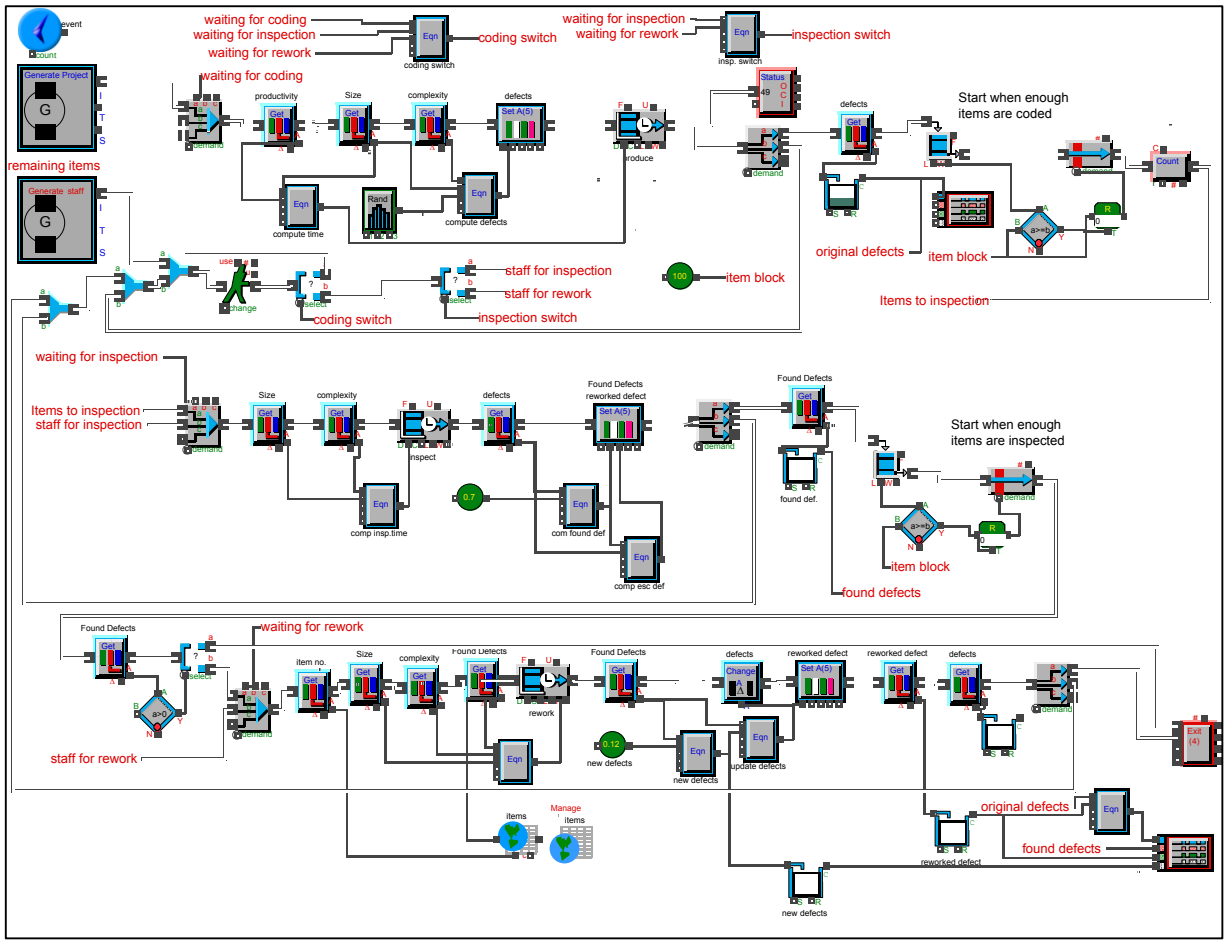


Figure 2.8 Discrete model of an inspection based process

In Figure 2.8 the discrete model of an inspection based process is shown. This model represents one phase in a software development process, the implementation of modules with subsequent inspection. The first 100 steps are used to generate the items, people and software modules. The upper line represents the implementation or production of the module, in the middle line the phases for finding and colleting the defects and in the lower line the rework of the defects found are modeled.

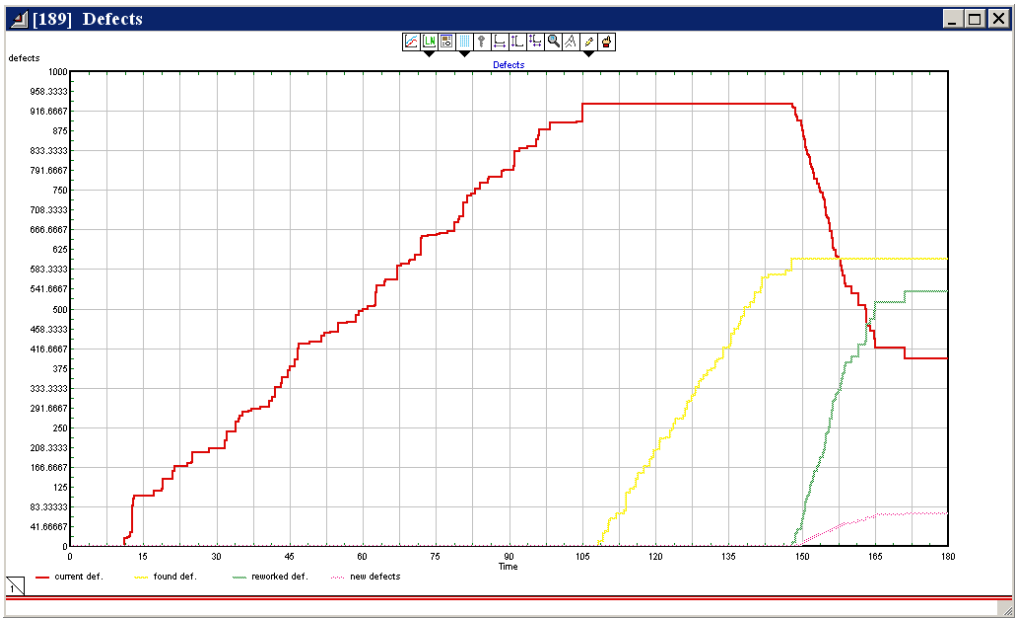


Figure 2.9 Current, found, reworked, and new defects

In Figure 2.9 the red line shows the sum of all defects in all modules. This is a theoretical value because in reality, the total number of defects is never known. The yellow line shows the sum of all identified defects, the green one all reworked defects and the pink one all new defects introduced during the rework.

In the following, we will discus the assumptions of the specific initial simulation model for software development, Figure 2.8. The model simulates the generation of software items (e.g., code fragments) including their inspection and the rework of found defects. The model is a discrete event simulation model constructed in EXTEND. Since some of the interactions of the different systems and variables have not been determined up to now (and can, possibly, never be determined in a deterministic sense), stochastic influences are used in the model.

The simulation consists of four subphases as discussed below. An item passes the subphases 1-3 sequentially. A "batch size" is used for defining how many items are passed en bloc from one phase to the next one. Items finished in a subphase are stored in a FIFO queue before being transferred to the next one.

The labor pool is initialized by a given number of staff members. Staff is assigned to items during each of the phases 1-3. A person is assigned with higher priorities to tasks at earlier projects phases.

Time is generally measured in hours.

Subphase 0: Project generation and staff generation

- A project consists of a fixed and a priori known number of items, e.g., 100.
- The size of an item is measured in function points (fp) and assumed to be uniformly distributed.
- The complexity of an item is assumed to be uniformly distributed.
- The staff consists of a fixed and a priori known number of persons, e.g., 10.
- The productivity (of workforce) is measured in function points/hour and is assumed to be uniformly distributed.

Subphase 1: Item and defect generation

- The number of defects of an item is generated as size * complexity * ran where ran is a [0,1]-uniformly distributed random number.
- The coding time (=delay) of an item is calculated by size/productivity.

Subphase 2: Inspection

- One or more persons are batched with an item to be inspected.
- The efficacy of inspection (portion of found defects) is set as a constant, for instance 0.7.
- The number of found defects of an item is efficacy * no. of defects
- The inspection time (=delay) of an item is calculated proportionally to its size * complexity.

Subphase 3: Rework

- If defects are detected for an item, it is subject to rework
- Only a fixed portion of found defects are repaired, i.e. some defects are overlooked or new defects are produced
- The rework time of an item is assumed to be proportional to its size * complexity * number of defects detected.

2.6.5 Potentials for Optimization

Preliminary experiments with the discrete event simulation model have shown some effects of model parameters and stochastic influences on the simulation results: the project time (and thus the costs) and the quality of the products (number of defects). For instance, it is obvious that an inspection process decreases the number of defects while it requires additional working time invoking costs. A trade-off analysis could lead to a simultaneous consideration of these effects. Additionally, one would have to consider more time and costs in subsequent phases, especially software testing for reducing defects, if inspections were not performed.

Another aspect already visible in the simple model is the effect of the batch size. If, for instance, one requires all items to be finished in one subphase before the next subphase starts, the overall project time is longer than in the case of a small batch size. The first case leads to longer, possibly unproductive waiting times of staff members.

This result emphasizes the importance of scheduling tasks. This problem could be solved by optimization methods in a more complex simulation model. Online and offline strategies for solving such a subproblem might be employed and integrated with the simulation model. The integration of the scheduling and task assignment problem also reflects the needs of a project manager.

2.6.6 Further Plans

Currently, the discrete model does not reflect the complexity of the real process during a single inspection. An improvement of the model requires refinement of the organizational processes and a better foundation with empirical data. Moreover, interfaces have to be defined for turning the model into a module for larger processes.

3 Dissemination of Results

3.1 Dissemination Activities

The objective of the dissemination activities is to promote the commercial and scientific exploitation of the project's results. The plan is expanded in two directions: towards marketing activities, in order to enhance the commercial potential of the project results, and towards the dissemination of the project's results in the scientific sector. As a first step towards enhancing the commercial potential, a business model was developed (see below). Conference and workshop participation as well as publishing of results in journals are means towards the dissemination in the scientific community. Additionally, a project web site will be created.

3.2 Business Model

The business model consists of four steps to transfer the know-how gained in the project into industry. In the first step, the model is built, and then it is reviewed by the customer to check whether it matches the requirements. In the third step, the model is initialized with data of the customer. In the last step, the model is used.

1.) Expert interview and building of the model
The first step consists of three tasks:
- Clarification of the problem and capturing the structural aspects of the process in an expert interview.
- Modeling of the software process.
- Use case diagram to capture the mental model.
- Simulation model of the process.

The involved persons are the simulation expert, the domain expert of the customer and the manager of the customer if he or she is interested in the results.

The results should include a process model and the use case diagram for the qualitative model

It will last approx. two months.

2.) Model review by the customer

In the second step, the customer reviews the models and the models will be revised.

The involved persons are mainly the simulation expert and the domain expert.

The result should be the accepted models.

This will last approx. one day up to one week.

3.) Initializing the model with data
Now the model is initialized with empirical data or with estimates from experts if empirical data is not available.

The involved persons are the simulation expert and the quality manager of the customer because he or she has to provide the data.

The result is a simulation model that can provide quantitative data.

It will take approx. two or three weeks to add the data into the model.

4.) Usage of the model
Testing of different fictive scenarios or constellations during a workshop.

The involved persons are the simulation expert, the process or domain expert of the customer, and the manager of the customer.

As a result, the customer gets the output of the simulation runs for the different scenarios and recommendations on how to change or optimize the process.

It will take approx. one day for the workshop and additional time to answer further questions of the customer.

3.2.1 Additional Service

The following services can be additionally supported:

- Performing a measurement program beforehand to document the current situation. This data can be used to initialize the model and to show the improvements.
- Guiding the process changes to improve the process.

4 Glossary

Activity (Synonym for →Actual Process)

An atomic or composite task in the real world that corresponds to a →process in the model world. An activity may contain other subactivities. Activities cover software development and maintenance activities as well as project management or quality assurance activities. An activity has precise starting and ending points.

Actual Process

The real process that is actually performed within an organization (see also →process types) [BFL+95].

Agent

A human or a →tool performing the →activities related to a →role. An agent is characterized by skills, cost and availability.

Artifact

Anything that is created, produced or modified in a project, either as a desired result of the project or as an intermittent result. An Artifact is part of the real world.

consume

Relationship between →process and →product: The consume relation identifies the products that are only used as input to a process and are not modified during the process.

Descriptive Software Process Model

A →process model describes how software is actually developed. [CKO92]

Desired Process

What the →process owner wants the →process to be (see also →process types) [BHK96]

Life cycle model

A lifecycle model is the management view of a →process model. All processes are structured in different phases (e.g., analysis, design, etc.) that contain the related processes. The time management of a project usually uses the life cycle model whereas the planning of the effort uses the process model.

Measure

The mapping of a value to an attribute of a process, product or resource.

Model

Abstract and simplified representation of a real-world object (→ system). A model is reduced to those aspects of the real world object modeled that are relevant (or its creator believes are relevant) for comprehension and for the intended use of the model. Generally we can distinguish between →product models, →process models, and →resource models.

modify

Relationship between a →process and a →product: This relationship identifies products that are changed within the process. These products are both input and output of the process.

Observed Process

What an →agent sees and understands from the actual process (see also process types) [BHK96].

Official Process

The formalization or documentation of the →desired process (see also process types) [BHK96].

Optimization

Identification and solving of an ➠ optimization problem.

Optimization method

Algorithm for solving an ➠ optimization problem.

Optimization problem

A set of state variables of a ➠ system described by restrictions together with an objective function. The solution consists of values for the state variables such that a different choice of values does not lead to improvement of the objective function.

Organization

An organization is an administrative or functional structure. Organizational units of human resources are important to analyze the →process. They facilitate understanding of communication channels or other social and hierarchical aspects. Their disadvantage is that they are very difficult to detect.

Parameters

Values for adjusting a ➠ simulation model that do not change during a simulation run.

Perceived Process

What an agent thinks the →process should be. The perceived process is based upon the understanding of the →official process (see also →process types) [BHK96].

Prescriptive Software Process Model

A prescriptive software process model describes how software should be developed (see also →descriptive software process model).

Process (Synonym for →Software Process)

A set of partially ordered steps intended to reach a goal [FH93]. The goal of a process is to produce or evolve software. For sake of simplicity, we will use the term process instead of software process in this context, unless explicitly stated. A process is an instance of a process model. A process is part of the model world.

Process Engineer

The process engineer's job is to elicit the process information from the process performers and use this knowledge to build a consistent and complete model of the process.

Process Instance (Synonym for →Process)

A process instance is the instantiation of a process model.

Process Model

An abstract representation of a software →process.[FH93] It describes a class of processes.

Process Modeling Language

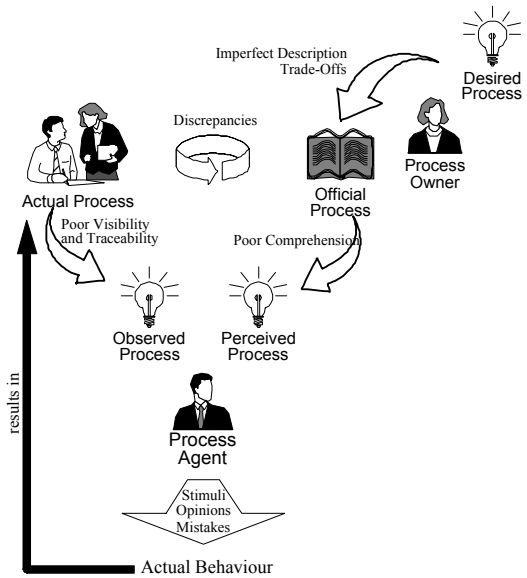
Formal notation used to describe software process models.

Process Owner

The person responsible for the software →process under consideration.

Process Types

(based upon [BHK96])



Product (Synonym for Product Instance)

Representation of an artifact of the real world (e.g., code component, requirements specification) in the model world. A Product is part of the model world. It is an instantiation of a →product model.

Product Instance (Synonym for →Product)

Product model

A product model describes the static properties of a class of →products and especially the structure.

Project

A project comprises all processes needed to develop and construct a system. Each project is characterized by specific goals and a specific development context.

Project plan

A project plan integrates instances of productmodels, process models and resource models for a specific project context.

Resource

A resource is the representation of an →agent in the model world. A resource is an instantiation of a →resource model.

Resource model

A resource model describes a class of →resources.

Role

A set of responsibilities, rights, and skills necessary to enact a specific →activity. A Role can be assumed by an agent.

Simulation

Reproduction of a ➠ system with its dynamic processes in an experimental ➠ simulation model, for obtaining results that can be transferred to reality. It is intended to reproduce the input-output-relationships of the considered system. The execution or solution of the simulation model is done with an appropriate ➠ simulation method.

Simulation, discrete

Characterized by a ➠ simulation method with a discrete time axis. The progress of time can be simulated by two approaches: by an event-oriented simulation (i.e. state transitions within the ➠ simulation model are caused by occuring events) and by time controlled simulation (i.e., the simulation time progresses

by an a priori given constant increment of time Δt. The state transitions within the previous interval of time Δt are calculated just after the incrementing of time. Choosing a very small value for Δt leads to an approximation of ➠ continuous simulation)

Simulation, continuous

Characterized by a ➠ simulation method where the time variable and all state variables for describing the model are continuous. The dynamics of the system are described by a set of coupled differential equations.

Simulation method

Defines the algorithm for calculating the temporal dynamics for the ➠ simulation. Depending on the system and the purpose of application ➠ continuous and ➠ discrete simulation are used for progressing the time within a model.

Simulation model

A ➠ model for purposes of simulation. It is characteristic for a simulation model that it can be used for experiments providing the possibility of a systematic variation of ➠ parameters.

Software Process (Synonym for →Process)

System

An excerpt from reality defined by the composition of objects related to each other, which can be considered as subsystems. Interactions between the components are, e.g., based on flows of energy, material, and information.

System, dynamic

A ➠ system that is characterized by its internal state comprising all state variables necessary for describing it at any point of time. The state of a component may influence the states of other components and its own succeeding states. In this way, a dynamic system is characterized by a memory effect.

Tool

A computer program supporting or automating an →activity or part of it.

References

[ABK+94] James W. Armitage, Lionel Briand, Marc I. Kellner, James W. Over, and Richard W. Phillips. Software process definition guide: Content of enactable software process representations. Special Report (Draft) CMU/SEI-94-SR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213-3890, December 1994. Not approved for public release.

[AK94] James W. Armitage and Marc I. Kellner. A conceptual schema for process definitions and models. In Dewayne E. Perry, editor, *Proceedings of the Third International Conference on the Software Process*, pages 153–165. IEEE Computer Society Press, October 1994.

[Attar] Attar Software: Reference Manual Xpert Rule Miner, Reference Manual XpertRule, http://www.attar.com.

[Bas97] Basilo V.R., *Evolving and Packaging Reading Technologies*, Journal of Systems and Software, 38(1).

[BC84] J. Banks, J. S. Carson, II: Discrete-event system simulation. Englewood Cliffs: Prentice-Hall 1984.

[BHK96] Lionel Briand, Dirk Höltje, and Hubert Kempter. Process modelling guidelines: Version 1.0. Technical report, Centre de recherche informatique de Montréal (CRIM) & Daimler–Benz AG, Montréal, Québec, Canada H3A 2N4, February 1996. Daimler–Benz Software Quality Project.

[BHL90] W. Busse von Colbe, P. Hamann, G. Laßmann: Betriebswirtschaftstheorie 2. Absatztheorie. 3rd. Ed. Springer, Berlin, 1990.

[Card99] Card, S.K., J.D. Mackinlay and B. Shneiderman, Eds, *Readings in Information Visualization: Using Vision to Think.* San Mateo, CA: Morgan Kaufmann 1999.

[ChSt00] A. M. Christie, M. J. Staley: Organizational and Social Simulation of a Software Requirements Development Process, Software Process Improvement and Practice 5, 103-110, 2000.

[CKO92] Bill Curtis, Marc I. Kellner, and Jim Over. Process modeling. *Communications of the ACM*, 35(9):75–90, September 1992.

[CFFS92] Reidar Conradi, Christer Fernström, Alfonso Fuggetta, and Robert Snowdon. Towards a Reference Framework for Process Concepts. In Jean-Claude Derniame, editor, *Proceedings of the Third European Workshop on Software Process Technology*, pages 3–17. Springer–Verlag, 1992.

[CFF93] Reidar Conradi, Christer Fernström, and Alfonso Fuggetta. A conceptual framework for evolving software processes. *ACM SIGSOFT Software Engineering Notes*, 18(4):26–35, October 1993.

[Dola00] P. Donzelli, G. Iazeolla: Hybrid Simulation Modelling of the Software Process, submitted to ProSim2000 Modeling Workshop, 2000.

[DrLu99] A. Drappa, J. Ludewig: Quantitative Modeling for the Interactive Simulation of Software Projects, Journal of Systems and Software, 46, 113-122, 1999.

[Dut93] James E. Dutton. Commonsense approach to process modeling. *IEEE Software*, 10:56–64, July 1993.

[EbS93] Robert G. Ebenau, Susan H. Strauss, *Software Inspection Process*, McGraw-Hill, Inc. 1993.

[FH93] Peter H. Feiler and Watts S. Humphrey. Software process development and enactment: Concepts and definitions. In *Proceedings of the Second International Conference on the Software Process*, pages 28–40. IEEE Computer Society Press, February 1993.

[Fo61] J. W. Forrester: Industrial Dynamics. Cambridge MA: Productivity Press 1961.

[Fo68] J. W. Forrester: Principles of Systems. Cambridge MA: Productivity Press 1968.

[Gonz96] Gonzalez, C., Does Animation in User Interfaces Improve Decision Making? In Proceedings of CHI'96 Conference on Human Factors in Computing Systems (Vancouver, Canada, Apr 13-18, 1996), 27-34.

[Gruh93] V. Gruhn: Software Process Simulation in Melmac, Systems Analysis Modelling Simulation 11, 121-141, 1993.

[IEE91] Institute of Electrical and Electronics Engineers. *IEEE Standard for Developing Software Life Cycle Processes*, 1992. IEEE Std. 1074-1991.

[Kaw91] Peter Kawalek. The process modelling cookbook: version 1. Technical report, University of Manchester and British Telecommunications, September 1991.

[KS93] J.R. Katzenbach and D.K. Smith. *The Wisdom of Teams*. Harvard Business School Press, 1993.

[LaD98] Oliver Laitenberger, Jean-Marc DeBaud, *An Encompassing Life-Cycle Centric Survey of Software Inspection*, Fraunhofer Institute for Experimental Software Engineering, Germany, ISERN-98-32, 1998.

[LAS97] C. Lin, T. Abdel-Hamid, J. Sherif: Software-Engineering Process Simulation Model (SEPS), Journal of Systems Software 38, 263-277, 1997.

[Lon92] Jacques Lonchamp. Supporting social interaction activities of software processes. In J. C. Derniame, editor, *Proceedings of the Second European Workshop on Software Process Technology*, Lecture Notes in Computer Science Nr. 635, pages 34–54. Springer–Verlag, September 1992.

[Lon93] Jaques Lonchamp. A structured conceptual and terminological framework for software process engineering. In *Proceedings of the Second International Conference on the Software Process*, pages 41–53. IEEE Computer Society Press, February 1993.

[Lusti99] M. Lusti: Data Warehousing and Data Mining – Eine Einführung in entscheidungsunterstützende Systeme, Springer Verlag, 1999.

[MaKh97] R. J. Madachy, B. Khoshnevis: Dynamic Simulation Modeling of an Inspection-Based Software Lifecycle Process, Simulation 69, 1, 35-47, 1997.

[MaTa00] R. J. Madachy, D. Tarbet: Case Studies in Software Process Modeling with System Dynamics, Software Process Improvement and Practice 5, 133-146, 2000.

[MaRa00] R. Martin, D. Raffo: A Model of the Software Development Process Using Both Continuous and Discrete Models, International Journal of Software Process Improvement and Practice 5, 2/3, 147-157, 2000.

[MeCo97] D. Merrill, J. Collofello: Improving Software Project Management Skills Using a Software Project Simulator, presented at Frontiers In Education Conference (FIE), 1997.

[Nauck96] D. Nauck, F. Klawonn, R. Kruse: Neuronale Netze und Fuzzy-Systeme- Grundlagen des Konnektionismus, Neuronaler Fuzzy Systeme und der Kopplung mit wissensbasierten Methoden, vieweg, 1996.

[PF91] L.C. Plunkett and R. Fournier. *Participative Management: Implementing Empowerment*. Wiley, 1991.

[PfLe99] D. Pfahl, K. Lebsanft: Integration of System Dynamics Modelling with Descriptive Process Modelling and Goal-Oriented Measurement, The Journal of Systems and Software 46, 135-150, 1999.

[PfLe00] D. Pfahl, K. Lebsanft: Knowledge Acquisition and Process Guidance for Building System Dynamics Simulation Models: An Experience Report from Software Industry, International Journal of Software Engineering and Knowledge Engineering 10, 4, 487-510, 2000.

[PKR00] D. Pfahl, M. Klemm, G. Ruhe: Using System Dynamics Simulation Models for Software Project Management Education and Training, presented at the Software Process Simulation Modeling Workshop (ProSim2000), London, 10-12 July 2000.

[PMD99] A. Powell, K. Mander, D. Brown: Strategies for Lifecycle Concurrency and Iteration - A System Dynamics Approach, The Journal of Systems and Software 46, 151-161, 1999.

[RCL98] Ioana Rus, James Collofello, Peter Lakey ; Software process simulation for reliability management; The Journal of Systems and Software 46, p. 173-182 ; 1999.

[RCL99] I. Rus, J. Collofello, P. Lakey: Software Process Simulation for Reliability Management, Journal of Systems and Software 46, 173-182, 1999.

[Rus97] I. Rus; Modeling the Impact on Project Cost and Schedule of Software Engineering Practices for Achieving and Assessing Software Quality Factors, Ph.D. Dissertation, Arizona State University, 1997.

[Spen96] Spenke, M., C. Beilken and T. Berlage, FOCUS: The Interactive Table for Product Comparison and Selection. In Proceedings of ACM Symposium on User Interface Software and Technology (Seattle, WA, Nov 6–8, 1996), 41–50.

[Ste00] John D. Sterman, Business Dynamics, Systems Thinking and Modeling for a Complex World, McGraw – Hill, 2000.

[Sw01] J. J. Swain: Power Tools for Visualization and Decision-Making OR/MS Today February 2001.

[Sw99] J. J. Swain: Imagine New Worlds. OR/MS Today, February 1999, 38-41.

[VM97] Martin Verlage, Jürgen Münch "Formalizing software engineering standards" In Proceedings of the 3rd International Symposium and Forum on Software Engineering Standards (ISESS '97). Walnut Creek, California, USA, March 1997.

[VBG+95] Martin Verlage, Christian Bunse, Peter Giese, Wolfram Petsch "Three Approaches for Formalizing Informal Process Descriptions" GI/GMA/IFIP/IFAC 5th International Workshop on Experience with the Management of Software Projects (MSP - 95), Karlsruhe, BR Deutschland, 27.-29. September 1995.

[Web84] Merriam Webster. *Webster's Ninth New Collegiate Dictionary*. Merriam-Webster, Springfield, Massachusetts, 1984.

[Weiß99] M.G. Weiß: Regulation Thermography and long term ECGs: Mathematics for Diagnosis Aiding in Medicine, International Congress of Industrial and Applied Mathematics ICIAM 99, Edinburgh, Edinburgh Press.

Document Information

Title:	Simulation-based Evaluation and Improvement of Software Development Processes
Date:	August 16, 2002
Report:	IESE-048.02/E
Status:	Final
Distribution:	Public