Creating a Code Inspection Model for Simulation-based Decision Support

Holger Neu, Thomas Hanne, Jürgen Münch, Stefan Nickel, Andreas Wirsen

Abstract— **Due to the fact that organizations developing software face ever increasing challenges to the quality, costs, and duration of software development, process models are used to understand, document, visualize, plan, and improve the development process. Usually, these models show the static structure of the processes, while the dynamic behavior is difficult to predict. Simulation models of software development processes can help to determine the process dynamics. While most of the simulation models proposed up to now are based on the system dynamics approach, we develop a discrete-event simulation model focusing on the inspection process that allows a more detailed representation of organizational issues, products and resources. In contrast to a system dynamics approach in a discrete-event simulation model, software products (such as code components) and resources (such as developers) are explicitly represented with attributes (e.g., size, skills).**

In this article we sketch the development of the model, its structure, and the results of selected experiments with the model. The model aims at supporting decision making for introducing and tailoring inspections. The question of which artifacts should be inspected and how inspection activities should be staffed will be discussed. The model considers influences of varying project and context conditions and can therefore be used as a basis for the adaptation to different development environments.

Index Terms—**Decision Support, Discrete-Event Simulation, Inspections, Process Optimization.**

I. INTRODUCTION

DECISION support for planning software development projects is a crucial success factor. The special characteristics (such as context-orientation, uncertainty, complexity, creativity, human-based processes) aggravate the planning of software development projects in contrast to the planning of many other production processes. The selection and tailoring of appropriate software processes, methods, and tools for the development of high quality software requires knowledge about their effects under varying project conditions. Up to now, selection has essentially relied on subjective knowledge, empirically gained experience from previous projects, or results from expensive controlled laboratory experiments. This leads to a situation where decisions concerning alternative processes and alternative implementations are only insufficiently supported.

In general, simulation can be used and already is being used in technical environments for planning a system a priori (e.g., before implementing it), for controlling it (e.g., for operative or online usage), or for analyzing it (a posteriori application). The reasons for creating a simulation model can be classified with respect to six categories [15]: strategic management, planning, control and operational management, process improvement and technology adoption, understanding, training and learning. The simulation model we have developed is intended to support the planning with respect to the following two aspects: On one hand, the variables of interest (e.g., total effort) can be predicted for a given process. On the other hand, simulation supports the decision on selecting alternative processes.

Before building a simulation model, the scope of the model has to be defined in accordance with the expected result variables, the process abstraction, and the input parameters. In general, the model scope usually fits one of the following points of view: portion of the life cycle, development project, multiple, concurrent projects, long-term product evolution or long-term organization. For the process abstraction, the model builder has to identify the key elements of the process, the relationships between these elements, and the behavior of these elements. Obviously, the relevant elements are those necessary for fulfilling the purpose of the model. Important elements to identify are, for instance, the key activities and tasks, objects (code units, designs, and reports), resources (staff, hardware), dependencies between activities and flow of objects, loops (iteration, feedback) and decision points, input parameters and result variables. To run a simulation model, the input parameters need to be initialized and the model has to be calibrated and validated with respect to the target organization. Validation can be done through reviews and inspections of the model. However, in order to make a model fit an organization, the input data has to come from the organization. The quality of the simulation results depends on the accuracy of the input data. In an industrial setting, the data is often not available because the

Manuscript received October 16, 2002. This work was supported in part by the German Bundesministerium für Bildung und Forschung (SEV Project) and the Stiftung Rheinland-Pfalz für Innovation (ProSim Project, no.: 559).

Holger Neu is with the Fraunhofer Institute Experimental Software Engineering, Sauerwiesen 6, 67661 Kaiserslautern, Germany (phone: +49-6301-707-155; fax: +49-6301-707-200; e-mail: Holger.Neu@iese.fraunhofer.de).

Jürgen Münch is with the Fraunhofer Institute for Experimental Software Engineering (IESE) in Kaiserslautern, Germany (e-mail: muench@iese.fraunhofer.de).

Thomas Hanne is with the Fraunhofer Institute for Industrial Mathematics (ITWM) in Kaiserslautern, Germany (e-mail: hanne@itwm.fhg.de).

Stefan Nickel is with the Fraunhofer Institute for Industrial Mathematics (ITWM) in Kaiserslautern, Germany (e-mail: nickel@itwm.fhg.de).

Andreas Wirsen is with the Fraunhofer Institute for Industrial Mathematics (ITWM) in Kaiserslautern, Germany (e-mail: wirsen@itwm.fhg.de).

measurement data needed was not captured or different measures were collected. Useful strategies for handling these situations can be found in [16]. Problems with the availability of data and also with the acceptance of simulation techniques when those are introduced are well known from other areas of application, but experience has shown that such difficulties can be overcome [17].

This paper discusses a discrete-event simulation model to support the planning of code inspections, which was developed within a larger research project [30]. Due to the context-orientation of software development (i.e., there is no unique approach for performing software development), a method for the fast and cost-effective creation of simulation models is needed. Hence, a corresponding method is sketched based on our modeling experience with the concrete inspection model. The paper is organized as follows: In Section 2, some related work is discussed. In Section 3, we sketch the method for building the discrete-event simulation model. Section 4 describes the model prototype and the rationales behind it. Section 5 presents results from applying the simulation model to the problem of appropriately selecting code items for inspections and determining the size of inspection teams. Finally, Section 6 gives an outlook on future work.

II. Related Work

A plethora of approaches has been proposed to support decision-making in software development (e.g., decision tables, expert systems, experience analysis). One promising approach is based upon the combination of process simulation and empirical data from real experiments and case studies. Rus et al. [18] describe the benefits in the following way: "(a) Simulation can use the empirical results from different contexts and apply them to a planning situation as appropriate. (b) The analysis of simulation results can point out situations and factors for which conducting empirical studies would be most worthwhile. Empirical studies about software inspections are an established discipline. A multitude of controlled experiments and case studies has been reported in the literature (e.g., [19]). Moreover, modeling and simulation are increasingly applied to software processes and broaden their understanding. Raffo et al. [20] describe the multifaceted relationships between empirical studies and the building, deployment and usage of process and simulation models. Several models for simulating inspections are described. They mainly differ with regard to the intended purpose (e.g., prediction, control), the dependent variables of interest (e.g., cycle time, reliability), the development phases considered (e.g., design, all phases), the simulation technique, and the degree of combining simulation with other techniques that support process understanding (e.g., descriptive process modeling, GQM). In the following, some essential contributions are sketched.

Rus et al. [21] present a process simulator for decision support that focuses on the impact of engineering practices on software reliability. The simulator consists of a system dynamics model and a discrete-event simulation model. The continuous model is intended to support project planning and predict the impact of management and reliability engineering decisions. The discrete-event model is better suited for supporting project controlling. One main purpose of the discrete-event model is to predict, track, and control software defects and failure throughout a specified period.

Madachy [22] sketches a system dynamics simulation model of an inspection-based life cycle process that demonstrates the effects of performing inspections or not performing them, the effectiveness of varied inspection policies, and the effects of other managerial decisions such as resource allocation. The model does not take into account schedule pressure effects and personnel mix.

Tvedt and Collofello [23] describe a system dynamics model aiming at decision support with regard to several process improvement alternatives. The dependent variable of interest is cycle time. The model is intended for understanding cause-effect relationships such as the influence of the implementation of inspections on cycle time reduction. The modeling approach distinguishes between a base model and several modular process improvement models (i.e., one for each improvement alternative).

Pfahl and Lebsanft [25] combine process simulation techniques with static modeling methods, namely software process modeling and measurement-based quantitative modeling. They propose the IMMoS approach that integrates system dynamics modeling with descriptive process modeling and goal-oriented measurement. The descriptive process model is used as a starting point for identifying causal relationships. Goal-oriented measurement is used for deriving measures from goals that are determined by the needs of a system dynamics model. Benefits of this combination are synergy effects from using already existing and proven methods and overcoming weaknesses of system dynamics model building.

In contrast to these contributions, the simulation model described in this paper is the first discrete one, focusing more strongly on the organizational and personal influence factors on inspections. It is mainly developed for decision support purposes.

Several papers discuss the steps towards building discrete-event simulation models. As an example, Raffo and Harrison [24] describe the creation of a discrete-event simulation model with the focus on integrating feedback from the software development process.

In contrast, the method sketched in this paper focuses on the integration with descriptive process modeling and goal-orientation as well as on integrating organizational issues of the development process. For modeling organizational issues, we need a greater level of detail. Discrete-event simulation models are more concrete than system dynamics models with regard to the static objects (tasks) and moving objects (work products) in the process. Also, it is possible to model individual persons and distinguish between work products [28].

III. Building the Model

In this Section, we explain the steps of building the discrete-event simulation model. Since many publications in the area of software processes modeling only describe simulation models and results of their application, our focus is now on explaining the steps of building a model. These steps are

based on our practical experience and will be refined in the future in order to define a method for the creation of discrete-event simulation models in a systematic procedure. This method is similar to the one proposed in [1], which considers the case of creating system dynamics models. We give a short summary of the steps that are explained later in more detail.

In the first step, the goal of the simulation has to be defined. Then, as a second step, the static process model, if one exists, has to be analyzed, otherwise a process model has to be created before. Step one and two can be performed in parallel. In the third step, we identify the factors that influence the interesting variables according to the goal of the simulation. In the fourth step, we determine the quantitative relationships required for the discrete-event simulation model. Although these steps seem to be performed in a sequential order, it is often necessary to go back to previous steps, i.e., a simulation model is usually developed in an iterative way.

A. Step 1: Definition of the Simulation Goal

If a simulation model is detailed enough, it can be used to satisfy various different goals. The effort for building such a detailed model, if possible at all, is far too high to be reasonable. Therefore, we have to reduce the model level of detail to address the most important goals of the simulation project, which can be identified by the GQM (Goal/Question/Metric) [8][9] method. GQM is usually applied for defining the measures to be collected during a project for purposes such as understanding, controlling, or improving. In our context, the defined metrics are not used for creating a measurement plan; instead, they can be used to identify possible input and output parameters.

The GQM goal definition consists of five dimensions, which describe the goals in a structured way.

Analyze the ***object of the study***
for a ***specific purpose***
with respect to a ***quality focus***
from a ***specific viewpoint***
in a ***specific context***.

Here we addressed the major goals, in order to determine the influence of organizational settings on (1) **effort**, (2) **duration,** and (3) **quality** by simulation. Because GQM goals should not cluster more than one purpose, one quality focus, and one viewpoint, the major goal should be divided into three GQM goal descriptions (e.g., analyze the inspection process for the purpose of decision-making with respect to effort from the viewpoint of a project planner in the context of company X).

The derivation of metrics with a GQM plan or the creation of GQM abstraction sheets can be used as a means for identifying independent variables (i.e., input variables for the simulation model) for a specific quality focus. The model granularity mainly depends on the viewpoint (i.e., a manager might be interested in a more abstract view than a developer) and the purpose (i.e., the expected results imply adequate model granularity). The object of the study might also influence the granularity (e.g., a process description cannot be further refined).

B. Step 2: Development of a Static Process Model

A static[1] process model describes the relationships between activities (processes), artifacts (products), roles and tools. The relationships between the activities and the artifacts are described in a product flow model, which shows the products used, produced and modified by an activity. The **roles** and **tools** are connected to the activities in an *involved* or *used* relation. A **role** is *involved* in an activity or a **tool** is *used* when performing that particular activity. Sometimes a control flow completes the process model. A product transition model can help to understand the order of transformations if many activities with a lot of transformations are within the scope of the model.

For creating or refining the static process model we use the elicitation-based approach as proposed in [4] and the process modeling tool SPEARMINT [10]. The graphic representation of the static process model can depict the flow structure of the model and eases the creation of flow logic in Step 4.

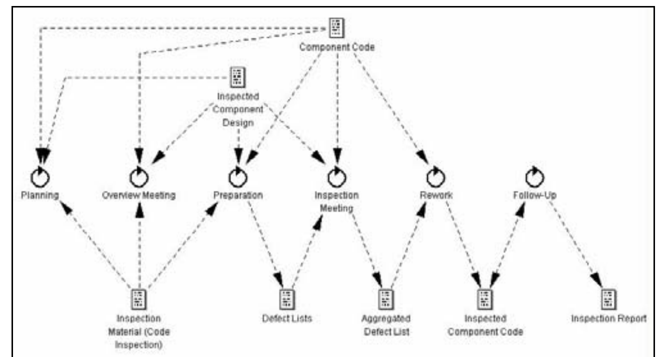

Fig. 1 Product flow of an inspection process according to [2].

In Fig. 1 the product flow of an inspection process is shown. The product flow includes the detailed activities of the inspection process and the artifacts that are being used, produced, and modified. It does not show explicitly the order of the activities performed during the inspection. If the order of activities is not obvious in the product flow diagram, it is possible to create a control flow diagram to show this order. Fig. 1 shows, for example, the inspection process, which is the refinement of the verification activity in the context of the coding activity. The activity *inspect component code* in Fig. 2 is the activity that contains the sub-process inspection.

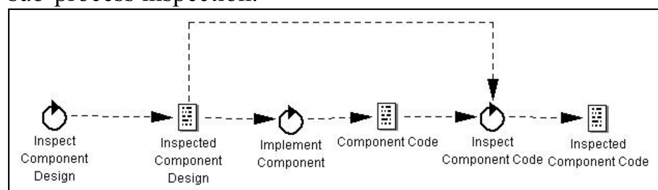

Fig. 2: The coding process with the component design as input and the inspected code as output parameter

In addition to the graphical representation, the static process model contains attributes that can be used to identify the variables of the qualitative model [1].

[1] A model is called static if there is no change in the model when the process is performed. A dynamic model changes over time and has a different state at every point in time.

C. Step 3: Development of a Qualitative Model

The qualitative model describes the relationships between the influencing factors and the relevant factors with respect to the goals. The determination of the influencing factors can be supported by the static process model, the experience of the modeler, the knowledge found in literature (e.g., results of laboratory studies), interviews with practitioners, and data mining. In many cases the factors are equal to those in other models and, therefore, the modeler can start with a basic set of factors.

The attributes from the static process model can be used to determine the variables for the dynamic model, e.g., attributes like lines of code (LOC) or defects, or times like the duration of an activity are supposed to be represented in the qualitative model.

A good visualization technique for qualitative models are cause-effect diagrams, causal-loop diagrams [3], or influence diagrams. They visualize the relations with arrows and the direction of the influence with plus or minus signs. A plus (minus) sign indicates that an increase in the influencing variable causes an increase (decrease) in the dependent variable. Cause-effect diagrams are also applied in developing system dynamics models, which are often used in software process modeling and simulation.

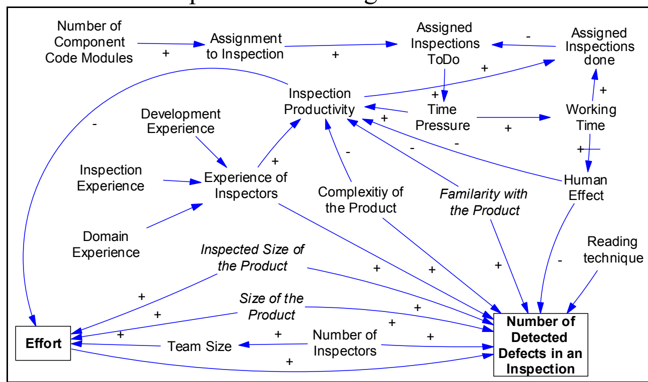

Fig. 3: A cause-effect diagram of an inspection process where the effort and the number of detected defects are the interesting factors.

Fig. 3 shows a cause-effect diagram of the two objective variables *effort* and *number of detected defects in an inspection*. Several of these factors are typical attributes of the artifacts, the activities, or the roles in the static process model that could be measured during the execution of the process. If empirical data is available from former projects, these data can be used in step four to determine the quantitative relationships for the model.

The measures identified during the goal definition in the beginning can aid in identifying variables that are not in the static process model, especially the behavior of the people performing the process.

D. Step 4: Development of a Quantitative Model

In the fourth step, the information collected in the first three steps is considered for quantifying the model. The GQM goal and plan can be used to derive the variable values that are set before or changed during the simulation. Also, the variables obtained as simulation results are specified by GQM. Usually, the input parameters are variables that are measured during a measurement program, as well as characteristics of the project, involved personnel, and the organization.

From the static model we know the sequence of steps, the decision points, and the activities where effort is consumed. The qualitative model depicts the input factors that influence the output factors and their dependencies. Note that the type of simulation model (discrete-event or system dynamics) pre-specifies extent and form of information to be quantified. On the other hand, the desired level of granularity and other goals related to the simulation determine the model type. Below, we focus on building a discrete-event simulation model because of its more detailed representation of organizational processes and persons and items related to them.

The simulation model is built by following the flow and control diagram and considering the related cause-effect relationships for each activity in the flow or control diagram. Both then define an activity block inside the simulation model with its related input and output variables.

Inside an activity block the relationships qualitatively described by the cause-effect diagrams have to be quantified by mathematical equations or some logical relationships. This is done by selecting one variable after another in the considered cause-effect diagram as an output variable, which then has to be explained by some variables it depends on. These input variables are predecessors in the cause-effect graph. Possible methods for quantifying the relationships between the output variable and the related input variables are expert interviews, pragmatic models, stochastic analysis, and data mining techniques.

Here one has to distinguish between the quantification of known relationships, i.e., exact linguistic descriptions that are available and have to be transformed into mathematical functions, and the generation of rules by applying data mining techniques to measurement data to describe relationships that are not obvious or were not considered up to now. The latter one is the case in simulating the software inspection process, for example.

The choice of the data mining techniques for rule generation depends, of course, on the data or information available, i.e., measurement data, linguistic descriptions, etc. The granularity of the model also determines the measurement data needed for rule generation, i.e., measurement data or information for all chosen variables are required. It is obvious that for building a discrete-event simulation model, more measurements are needed than for building a system dynamics model, since the former one includes many more variables. In a discrete-event simulation approach we also have to keep in mind that variables might have different states for similar objects, i.e., developers have different coding or inspection skills.

The techniques we considered up to now for the quantification of the qualitatively known relationships described in the cause-effect diagrams are neural networks and classification trees.

The neural networks used are feed forward neural networks with one hidden layer and a single output [5][6]. The corresponding network function is the mathematical equation describing the input-output relationship for the considered node in the cause-effect diagram and can be

plugged into the simulation model. The network function includes input-output weights for each unit in the hidden layer, which have to be determined. In general the weights are estimated from industrial data. The industrial data is split into a training set and a validation set. The training set is then used to estimate the weights of the network function, and the validation set is used to determine the quality of the resulting network function, i.e., the input variables were given values from the validation set. The network function is then evaluated for the given input values and the analytical calculated output is compared with the corresponding measurement data.

The problem, which often occurs especially in the context of the software development process, is that measurement data is not available for all input variables . In this case one has to make further assumptions or skip these variables.

Relevance measures [11], for example the partial derivatives of the network function with respect to the input variable, can help to determine the impact of each explaining (input) variable with respect to the explained (output) variable. By considering the validation results and the corresponding relevance measure, one can easily verify whether the estimated functional dependencies describe the input/output relation in a sufficient manner, whether the impact of a skipped variable is too large or whether an explaining (input) variable is missing. In the latter case, the missing variable should be determined, e.g., through a case-based-reasoning method [14], and a new rule has to be generated for it by considering more possible input variables than considered in the qualitative model or applying other data mining techniques. If a variable has very small relevance across its whole parameter range, it is redundant and does not explain the considered output variable. Thus, a relevance measure can also be used to validate the qualitative description of the dependencies given in the cause-effect diagrams.

The second technique applied for the quantification of the relationships we used are classification trees determined by the software tool XpertRule Miner [7]. Based on the information gain technique, a classification tree for an explained variable is calculated on the data set. The tree can be read from root to a leaf as a logical if-then rule for the input-output relationship. A leaf of the tree contains information about the percentage, variance, mean and standard deviation for the explained variable when applying the corresponding rule. Thus, the root of the tree denotes the variable with the greatest impact with respect to the explained variable. The splitting criterion used is based on the normalized standard deviation [7].

The relationships developed in this step will be used when the model is created and the equations are implemented in the model. The input parameters for the equations have to be provided with the input parameters for the model.

Currently, we are analyzing data on historical software development processes coming from two large companies. Unfortunately, these data (which were not collected for the purpose of fitting a simulation model) cover only some variables required for the discrete-event simulation model presented here. For instance, information on the assignment of tasks to persons and individual working times is missing. On the other hand, the data differ essentially from company to company, e.g., because of different organizational settings and measurement techniques.

Therefore, in the model discussed below we do not use complex functional relationships (as derived, for instance, from neural networks), but more simple functions as typical in the literature, which were adapted using the available industrial data and data from the literature. However, for application in industrial settings the more complex functional relationships should be used, since this allows a detailed adaptation to a company with its specific environment settings. Nevertheless, building and presenting the model is the first step in convincing software developers to collect the required data, which will then be used for model adaptation and refinement.

IV. Building a Discrete-Event Simulation Model

In the following, we describe the discrete-event simulation model, which is being developed as a prototype decision support tool for planning inspection processes. For building the model, we used the simulation tool Extend [12], which allows to build both discrete-event and continuous simulation models.

A. High-level Architecture of the Model

The macro structure of the simulation model reflects the sequence of tasks or sub-processes (such as design, coding, inspection, and unit test) according to the process diagram (Fig. 2). Similar to physical items in material flow systems, the software items are moved through these sub-processes of a typical software development process. Therefore, design and code documents (items) are represented by moving units (MUs) while static objects represent the tasks. The most important object for representing a task is an activity block. Such a block stores an item for the duration or working time of the corresponding activity and, thus, represents the temporal structure of a project.

Besides the duration, each task affects the quality of the processed item. In general, we assume that the quality of an item (design document, code document) is measured by its number of defects. Thus, during design[2] and coding, items with a specific size and number of defects are created. During inspection and test, some of the defects are found. During rework, found defects are removed (and possibly some new defects are produced).

For all activities represented by the model, we assume that their results are determined by attributes of the processed item, attributes of the person who performs the tasks, and organizational factors as qualitatively described in the related cause-effect diagram. For considering effects specific to the person assigned to the task, developers are represented by MUs just like items. In Extend, the linking of items and persons prior to a specific task is done by a block creating a compound MU, which represents an item together with an assigned person. Up to now, one main assumption of the model is that the assignment of tasks to persons is done in an arbitrary way or, more precisely, persons are

[2] In the following, especially in the simulation experiments discussed in Subsection D, we only consider the planning of code inspections that are assumed to be modeled in a more detailed and reliable way.

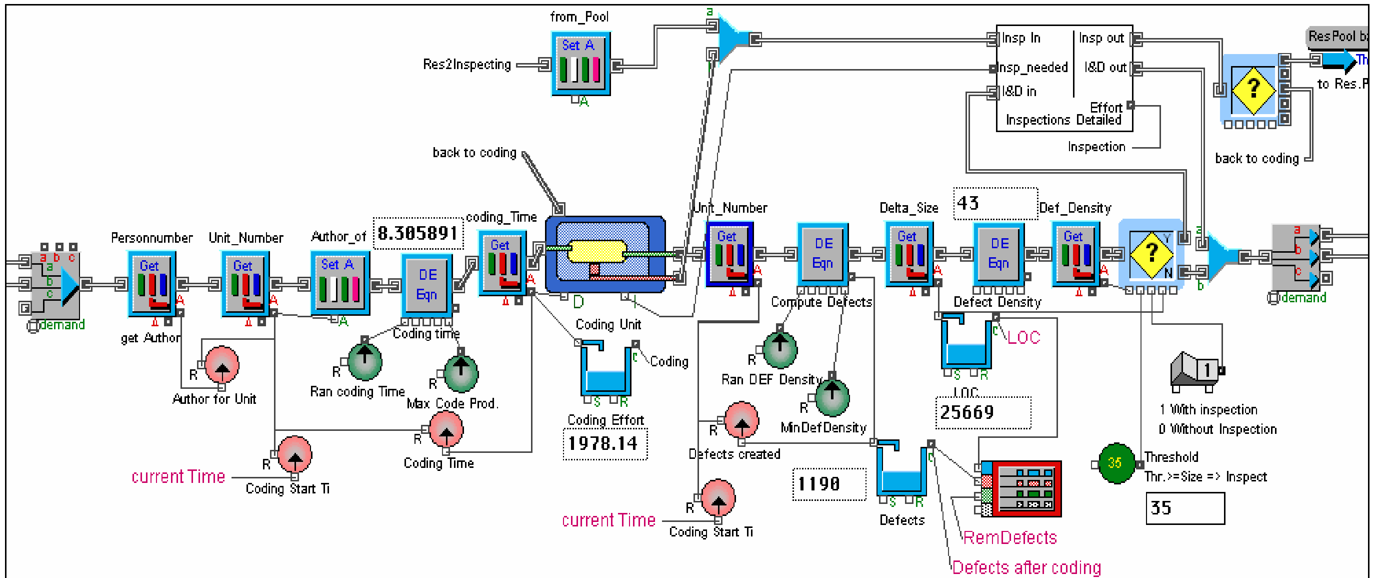

Fig. 4. Excerpt from the Extend visual Interface of the model. Coding with the inspection of selected code items. Items pass the main line from left to right. The first block on the left is a batch block for combining code modules and persons; the last on the right separates the batched modules and people.

selected from a staff pool in a "first come, first serve" (fcfs) fashion. This means that at the start of a simulation run items are batched with persons according to a given order (determined by item and person numbers) as long as persons are available. Later on, items waiting for their next processing steps are batched with the persons becoming available next.

Unlike the other tasks, inspections are assumed to be done in an interrupting way such that persons involved in coding or testing may interrupt their tasks and read the documents to be inspected in between. Thus, waiting times for building an inspection team can be avoided in the model.

After accomplishing a task, linked items and persons are un-batched. Persons are then directed back to the pool from where they are assigned and sent to new tasks. Items are directed to the subsequent sub-process, e.g., from design to coding or from inspection to rework. In some cases, there are alternatives for routing an item. For instance, rules may be applied for deciding on whether an item is subject to inspection or not. Moreover, switches can be used for activating or deactivating some parts of the model, e.g., the inspections, the design, or the testing and rework activity. In general, the connections of processes and sub-processes and the routing logic for the MUs should represent organizational rules of a considered real-life company.

Most of the Extend blocks in Fig. 4 are for accessing attributes or variables and for calculating and assigning new values. For instance, in the coding sub-process, the number of produced defects is calculated; in the inspection and test processes, the number of found defects is calculated, and in the rework processes, the number of defects is updated (considering new defects produced during rework).

The general sequence of blocks representing one sub-process serves the following purposes: First, the working time of the activity has to be calculated. Then, a compound item enters the activity block and stays there for that time. After that, its number of defects is updated.

The most important issue of each sub-process is to represent the quantitative relationships of the model in a valid way. This especially concerns the outputs of an activity, i.e., its consumed time and the effects on the number of defects. In general, these values are determined by attributes of the items, by attributes of the persons, and by general or project-specific factors. For some of the relevant data it is hardly possible to determine the necessary information in real-life projects. For instance, details on the specific experiences, skills, and productivities of persons are usually not available. Therefore, we are elaborating approaches for taking such human attributes into account, which are not directly observable, and for considering them in the quantitative logic of the model.

B. *Quantitative Relationships of the Model*

Following the guideline sketched in the previous section, we now explain some quantitative relationships that determine intermediary and output variables. Some of the variables used in the quantitative relationships are related to a code item i, or a person j, others are general parameters of the model.

1) *Parameters and Assumptions*

For an item i, we assume that its size, $size_i$, (measured, e.g., by the number of lines of code) and its difficulty or complexity, $cplx_i$, are the most important factors for the results of an activity. The complexity $cplx_i$ is assumed to be an adjusting factor distributed around 1, which serves for weighting the size according to the specific difficulty of processing the item.

For a person j, we assume that specific skills determine his or her quality of work and his or her productivity (work load per time unit) and, thus, the change in the number of defects and the time needed for performing a task. The specific skill values of the model are the coding quality skill (cqs_j), the preparation (inspection) quality skill (pqs_j), the coding productivity skill (cps_j), preparation productivity skill (pps_j), and the testing productivity skill (tps_j).

For ease of use we assume these skill values to be calibrated on a nondimensional [0,1] interval. Moreover,

working with such skill values instead of personal productivities allows an easier application of a learning model such as, for instance, [31]. A skill value of about 0.5 is typical for an average developer, while a value near 1 characterizes an experienced or high performance developer. Multiplied with given values for maximum quality of work or a maximum productivity (corresponding to skill values of 1), a person's actual defect (production, detection) rate and productivity can be determined. Thus, the following model parameters with respect to productivities are used: a maximum coding productivity, *mcp*, a maximum preparation productivity, *mpp*, and a maximum testing productivity, *mtp*.

With respect to defects, the following model parameters are used: The number of defects in relation to the size of the document to be produced is expressed by a minimum defect density, *mdd*. For rework, it is assumed that all found defects are removed but some of them not correctly, or that new defects are produced in a proportional relationship to the correctly removed ones. For expressing the ratio of new defects with respect to the removed ones, a rework defects factor *rdf* with $rdf < 1$ is used. For the unit test, a defect find rate, *dfr*, is applied. Note that all these model parameters are specific to various characteristics of a software development project, which are not explicitly considered in the model, e.g., the programming language and development tools.

2) Defect generation, detection, and rework

For the activity coding the number of defects produced during coding, pd_i, is calculated by:

$$pd_i = size_i \cdot cplx_i \cdot mdd / cqs_j \tag{1}$$

For the number of found defects during an inspection we consider the underlying cause-effect diagram (Fig. 3) to show the development of the equation for the defects found during an inspection for one item fdi_i. Thus fdi_i is the output variable and all other nodes connected with this variable serve as the explaining (input) variables.

$$fdi_i = (1 - \exp(tef_j \cdot its_i))gef \cdot pd_i \ \ /cplx_i \tag{2}$$

Note that not all influencing factor from the cause-effect diagram (Fig. 3) appear in this formula, some of them are omitted because of a lack of data and measurement problems, e.g., familiarity with the product. Other factors, e.g., size, are indirectly considered. (The size is used for calculating the number of produced defects pd_i .)

tef_i is a team effectiveness factor depending on the skills of the individual team members and other factors. A value of $tef_i = 3/7$, for instance, corresponds to a situation where 7 inspectors find 95% of the defects. For the inspection team, the number of found defects is assumed to be smaller than the sum of individually found defects (because of double counting). Therefore, the team size its_i is in a degressively increasing relationship to the number of found defects.

gef is a general effectiveness factor that reflects, for instance, the effectiveness of the chosen inspection or reading technique.

The new defects produced during rework are calculated by

$$rd_i = rdf \cdot fd_i \tag{3}$$

where fd_i is the number of found defects during the previous inspection or test.

Additionally defects are also found during testing. We assume that these are in an exponential relationship to the testing time tt_i [32] as follows:

$$fdt_i = d_i\left(1 - \exp\left(-dfr \cdot tps_j \cdot tt_i\right)\right), \tag{4}$$

where d_i are the defects remaining after coding, inspection, and rework. Alternatively, the number of to be found defects (or a desired defect density) may be pre-specified. In that case the above formula may be transformed for calculating the required testing time. The effects of rework after test are calculated in the same way as those of rework after inspection.

3) Effort

Based on the above assumptions, the relationships for determining processing times of activities (performed by person *j* on item *i*) and their effects on the number of defects can be expressed as follows. For the coding time, ct_i:

$$ct_i = size_i \cdot cplx_i / (mcp \cdot cps_j). \tag{5}$$

The individual preparation time, pt_{ij}, of an inspector is calculated as follows:

$$pt_{ij} = size_i \cdot cplx_i / (mpp \cdot pps_j). \tag{6}$$

The size can be the size of the actually inspected code if existing code is changed, or the total size for a new development. The rework activities are assumed to be closely connected to the coding activities. Thus, the same skill values are also used for determining the rework time. The relationship between rework activities (measured by the number of defects to be removed) and coding activities (measured by the size of the item to be produced) is expressed by an artificial factor, the average defect size, *ads*. The time for rework is then calculated by:

$$rt_i = fd_i \cdot cplx_i / (ads \cdot mcp \cdot cps_j). \tag{7}$$

The overall effort is computed by summing up all effort data that are calculated using the time needed for the activities of item i.

4) Duration

The goal variable duration results for each simulation run when the starting and finishing times of all activities, i.e., the project schedule, are fixed.

C. Other Aspects of Modeling

Due to the chosen granularity defined in the goal definition and other assumptions (see IV.B.1), not all details of the real-life software development process are modeled. For instance, meetings prior to an inspection are not represented in the model. Let us note that the sub-processes outside our model focus are modeled in a rougher way. Similarly, aspects outside the considered project are neglected, for instance the involvement of persons in other tasks or different projects. It would, in principle, be possible to have a pre-specified timetable for each person defining their availability for the current project.

Most human factors except productivity and quality skills for coding and inspection are neglected up to now. An extension of the model for representing effects of time pressure is under development. The organizational factors are expressed in the values for maximum productivities, etc., which are assumed to be fixed for the given project.

For setting these values of the given model, real-life data from industrial software development processes has been used. This data is not available up to now and we are also looking for data from the fields of psychology and manpower studies.

Even after careful refinement of the quantitative relationships, there are essential effects in a software development process that cannot be fully determined a priori, for instance, human effects such as fatigue, boredom, and other physical and mental factors. These human effects are considered by having stochastic elements in the process, which influence, for instance, the working times and numbers of defects produced. Therefore, the results of an activity with respect to the changed number of defects and the time needed for performing it are considered as random. For this reason, the above values are multiplied by random factors, which are assumed to be stochastically determined according to a lognormal distribution with an expected value 1. Multiple runs of the model can be used for estimating the effects of such random influences on the distribution of the model outputs, especially with respect to the objectives quality, project duration, and costs. Using such information, a project manager may get a better feeling for the risks within a scheduled project.

Another key aspect of real-life projects not explicitly treated is that of scheduling. The implicit fcfs task assignment and scheduling leads to sub-optimal results and can be improved as worked out in [29].

D. *Application of the Model*

As input data, the simulation model requires a specification of a software development project. Roughly said, such a project consists of item-specific data, person-specific data, and project-specific or general data. For each item to be produced (e.g., the source code of a module), the item-specific data includes the number of lines of code (or new/changed lines of code in case the item existed before) and a complexity measure. The person-specific data consists of estimates for quality and productivity skills of all members of the development team. General or project-specific data are, for instance, maximum productivity values (see above) or person costs per hour.

For an application of the model within an industrial context, these input data should be determined from earlier projects. For an application of the model unrelated to a real-life project, e.g., for educational purposes, it is possible to generate a stack of tasks (items) and a pool of persons randomly. Input and output data are stored in a text file linked via SDI Interface with the Extend simulation software [13]. An internal DB stores values for used distributions that can be changed if the model is to be fitted for a specific context.

After starting the simulation it is possible to get some information on the progress by switching on the animation. The value of this information is, however, rather marginal. More useful tools for keeping the user informed about the dynamics of the model are plotter blocks, which show the charts of specified variables. Additionally, several variable values are displayed within the visual interface of the simulation model.

V. Experiments with the Model

In this Section, we use the simulation model for two experiment series on variants of the software development process. In both series, here denoted by A and B, the objectives ‘duration’ and ‘overall effort’ are considered. For facilitating comparisons, the third objective, product quality (i.e., the number of final defects), is assumed to be constant. This is achieved by requiring the test phase to continue until the desired level of the defect density is reached. This means that products with more defects entering the test phase require more test and rework effort.

For the sub-processes testing and rework, it is calculated how much time the test activity requires to get a desired defects density. Thus, the rework effort depends on the number of defects to be found in tests, and after testing, the resulting number of remaining defects is always the same.

In A, we analyze whether inspections of all or selected items are useful compared to a software development process without inspections. In B, we analyze the question of what an optimal size of an inspection team might be.

Of course, there are further parameters of the simulation model that influence the effectiveness of inspections and that could be analyzed by the simulation model, e.g., the inspection technique. Corresponding studies based on the simulation model will be performed in the future.

A. *The Selection of Items for Inspection*

One of the key questions in introducing and planning inspections concerns the selection of items to be inspected. In order to compare different policies, we consider a project for producing software (creating new features for an existing product) with 100 items of different size, with 20 developers, and compare the overall effort and time spent for a specific defect density. We analyze three variants of a software development process: a) without any inspections, b) with inspecting all items, and c) with inspecting all items with a defect density larger than 35 defects per KLOC. This defect density threshold turned out to be reasonable according to the given defect distribution. This assumption can give a baseline for the effects of an optimal selection of code units for the inspection. Usually the number of defects in a piece of code is not known. The assumption of knowing the defect density can, however, give an upper bound estimate for the effects of an optimal selection of code units for the inspection. Other rules will be tested in the near future. For instance, based on measurement data, we have developed a classification tree. Following the tree in an if-then rule results in a single leaf for each code document giving the expected mean, the variance and standard deviation for the defects of this document. Now, for example, documents with an expected mean for the defects larger than a threshold could be inspected.

TABLE 1.
AVERAGE RESULTS OF 20 SIMULATION RUNS

Inspection strategy	Size of all items	Initial defects	Defects found in inspections	Defects after inspections and rework	Final defects	Overall effort	Process duration
No inspections	25669	1129	0	0	33	11567,57	880,56
All inspected	25669	1149	573	630	33	11062,94	764,07
Select item for inspection	25669	1155	474	507	33	11177,16	750,76

Table 1 shows the simulation results for 100 items with a purposed defect density of 1.5 defects per 1000 lines of code. Note that the differences in the initial defects result from the stochastic nature of defect generation. The model shows that the introduction of inspections increases the effort spent for the coding phase, but if the inspections are executed, the effort spent for testing and rework is reduced.

The overall effort is less for the simulation runs with inspections. Also, the duration of the project is shorter if inspections are executed.

These results are in accordance with the one found in literature [27], which suggests that the effort spent for inspections is less than the effort saved for testing. The results for the simulation with selecting items for inspections show a slightly shorter duration but a higher overall effort. As observed in the simulation experiments, the reason for that behavior is the scheduling of tasks. Large error-prone modules take a long time for testing and rework and, if started late with these modules the duration is prolonged. Here scheduling and optimization algorithms [29] can help solvethese questions.

As an alternative to the used testing policy, it would be possible to specify a time frame for how long a code item is tested (instead of specifying a desired defect density) so that the effort spent on testing could be kept constant. In that case, software development processes with inspections would result in better product quality at the end of the software development process.

B. *The Influence of the Team Size*

The number of found defects and the effort in the overall process (especially coding and test) depend on the number of inspectors involved in the inspection of one code item. For analyzing these effects we perform simulation runs of the model with the size of the inspection team varying from 1 to 10.

In Fig. 5 the graph shows the overall effort needed for the process. It significantly increases for more than four inspectors. Similar increases are shown for the duration for the overall process. If we consider the effort and the duration, the optimal number of inspectors is between two and four.

The other two lines in the graph show the number of defects found and missed during the inspection. Here we can see that increasing the number of inspectors does increase the number of defects found, but only digressively. With more than seven inspectors the number of found defects does not increase significantly. Therefore, an inspection team size of more than seven inspectors is not useful for increasing product quality.

As stated in [27], increasing the number of reviewers has a ceiling effect because the probability that defects are found by two or more inspectors increases with the number of inspectors. Therefore, adding inspectors does not increase the number of defects detected significantly and mainly increases the effort and time spent.

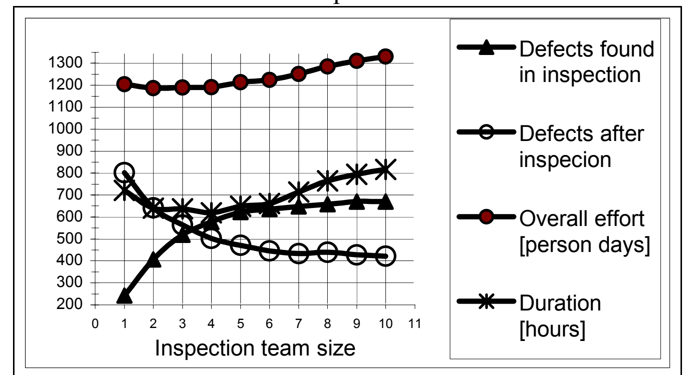

Fig. 5 Defects, duration and effort with respect to different numbers of inspectors (average values for 20 simulation runs).

Note that this example application of the model prototype does not reflect the situation of a specific real-life company, since adequate data for an accurate description of the quantitative relationships were not available. Such an adaptation of the model to the situation of a specific industrial company is planned for the near future.

VI. CONCLUSIONS

In this article we presented an approach on how to create a discrete-event simulation model for code inspection processes in a systematic way. It is important to have sound experience in the field of software engineering and simulation. It is expected that the approach can help to create the model faster with less iterations. The model for planning inspections we presented clearly shows the effects of organizational changes. Since developing software is a creative and human-based activity, we included the skills and also some stochastic factors for non-predictable influences.

The results of a simulation can show the influence of decisions during project planning and execution of actual or future projects. However, because of scarce data for determining the quantitative relationship and conceptional problems with the model validity (scope, etc.), the obtained results must be treated with caution. The model we presented will be extended in the future with a simple learning sub-model that considers the increase of knowledge and familiarity with the product during the execution of one

or several tasks. We believe that over several projects the skill of the inspectors is increasing and has an influence of the overall performance in future projects.

The future work will be to include the learning sub-model, connect the model to an ODCB database, and develop a user-friendly interface. We also want to test and refine our approach for creating a discrete-event simulation model. It is planned to create a more sophisticated test model.

ACKNOWLEDGMENT

We would like to thank especially Ioana Rus from the Fraunhofer Center for Experimental Software Engineering, Maryland for her useful comments on the paper and our work. Also we thank Sonnhild Namingha from the Fraunhofer Institute Experimental Software Engineering for reviewing the first version of the article.

REFERENCES

[1] D. Pfahl: "An Integrated Approach to Simulation-Based Learning in Support of Strategic and Project Management in Software Organisations", Stuttgart: Fraunhofer IRB Verlag, 2001.

[2] R.G. Ebenau, S.H. Strauss: "Software Inspection Process", New York: McGraw-Hill, Inc., 1994.

[3] J. D. Sterman: "Busines Dynamics – Systems Thinking and Modeling for a Complex World", Irwin McGraw-Hill, 2000.

[4] U. Becker-Kornstaedt: "Towards Systematic Knowledge Elicitation for Descriptive Software Process Modeling", In F. Bomarius and S. Komi-Sirviö, editors, Proceedings of the Third International Conference on Product–Focused Software Processes Improvement (PROFES), Lecture Notes in Computer Science 2188, pages 312–325, Kaiserslautern, September 2001. Springer.

[5] H. White: Learning in artificial neural networks: A statistical perspective. Neural Computation 1, 1989, 425-464.

[6] H. White: Connectionist nonparametric regression: multi layer feed forward networks can learn arbitrary mappings. Neural Networks 3, 1990, 535-549.

[7] XpertRule Miner, Attar Software GmbH: www.attar.com.

[8] R. van Solingen, E. Berghout: "The Goal/Question/Metric Method: a practical guide for quality improvement of software development", London: McGraw Hill, inc., 1999.

[9] L.C. Brinad, C. Differding, H.D. Rombach: "Practical guidelines for measurement-based process improvement", Fraunhofer Institute for Experimental Software Engineering, Germany, ISERN-96-05, 1996

[10] Spearmint: http://www.iese.fhg.de/Spearmint_EPG/

[11] A. Sarishvili: "Neural Network Based Lag Selection for Multivariate Time Series", PhD. Thesis, University of Kaiserslautern, 2002.

[12] D. Krahl: "The Extend simulation environment", J.A. Joines, R. R. Barton, K. Kang, P. A. Fishwick (Eds.): Proceedings of the 2000 Winter Simulation Conference. IEEE Press, 2000, 280-289.

[13] A. J. Siprelle R. A. Phelps M. M.Barnes: "SDI INDUSTRY: AN EXTEND-BASED TOOL FOR CONTINUOUS AND HIGH-SPEED MANUFACTURING", Proceedings of the 30th conference on Winter simulation, 1998, Washington, D.C., United States, Pages: 349 - 358

[14] R. Bergmann, S. Breen, M. Göker, M. Manago, S. Wess: "Developing Industrial Case-Based Reasoning Applications", Lecture Notes in Artificial Intelligence, Subseries of Lecture Notes in Computer Science, Springer, 1998

[15] M. I. Kellner, R. J. Madachy, D. M. Raffo: "Software process simulation modeling: Why? What? How?", Journal of Systems and Software 46, 2-3, 1999, 91-105.

[16] M. Kellner, D. Raffo: "Measurement issues in quantitative simulations of process models", Proceedings of the Workshop on Process Modelling and Empirical Studies of Software Evolution (in conjunction with the 19th International Conference on Software Engineering), Boston, Massachusetts, May 18, 1997. 33-37.

[17] F. McGuire: "Simulation in healthcare", J. Banks (Ed.): Handbook of Simulation.Wiley, New York 1998, 605-627.

[18] I. Rus, S. Biffel, M. Halling: "Systematically Combining Process Simulation and Empirical Data in Support of Decision Analysis in Software Development", SEKE 2002

[19] O. Laitenberger, J.-M. DeBaud: "An encompassing life-cycle centric survey of software inspection", Journal of Systems and Software 50, 1, 2000, 5-31.

[20] D. Raffo, T. Kaltio, D. Partridge, K. Phalp, J. F. Ramil: "Empirical studies applied to software process models", International Journal on Empirical Software Engineering 4, 4, 1999, 351-367.

[21] I. Rus, J. Collofello, P. Lakey: "Software process simulation for reliability management", Journal of Systems and Software 46, 2-3, 1999, 173-182.

[22] R. J. Madachy: "System dynamics modeling of an inspection-based process", Proceedings of the Eighteenth International Conference on Software Engineering, IEEE Computer Society Press, Berlin, Germany, March 1996, 376-386.

[23] J. D. Tvedt, J. S. Collofello: "Evaluating the effectiveness of process improvements on software development cycle time via system dynamics modelling", Proceedings of the Computer Software and Applications Conference (CompSAC'95), 1995, 318- 325.

[24] D. Raffo, W. Harrison: "Combining Process Feedback with Discrete Event Simulation Models to Support Software Project Management", Workshop on Feedback and Evolution in Software and Business Processes, London, UK, July 2000.

[25] D. Pfahl, K. Lebsanft: "Integration of system dynamics modelling with descriptive process modelling and goal-oriented measurement", The Journal of Systems and Software 46, 1999, 135-150.

[26] D.P. Freedman; G.M. Weinberg: "Handbook of Walkthroughs, Inspections, and Technical Reviews. Evaluating Programs, Projects, and Products", New York: Dorset House Publishing, 1990.

[27] O. Laitenberger, M. Leszak, D. Stoll, K. El Emam: "Quantative Modeling of Software Reviews in an Industrial Setting", 6th International Software Metrics Symposium. Metrics'99 - Proceedings (1999), 312-322 : Ill., Lit.

[28] H. Neu, T. Hanne, J. Münch, S. Nickel, A.Wirsen: "Simulation Based Risk Reduction for Planning Inspections", Proceedings of 4th International Conference on Product Focused Software Process Improvement, PROFES 2002; Rovaniemi, Finland, December 2002, 78-93

[29] T. Hanne, S. Nickel: "A multi-objective evolutionary algorithm for scheduling and inspection planning in software development projects". Report of the Fraunhofer ITWM 42, 2003.

[30] Project "Simulation-based Evaluation and Improvement of Software Development Processes". Web site at: www.sev-fraunhofer.de

[31] T. Devezas: "Learning Dynamics of Technological Progress". Paper presented at the 2nd International Conference on Sociocybernetics, June 26 – July 2, 2000, Panticosa, Spain

[32] G. M. Weinberg: „Quality Software Management. Volume 1. Systems Thinking". New York: Dorset House Publishing, 1991.