Using Empirical Knowledge from Replicated Experiments for Software Process Simulation: A Practical Example

Jürgen Münch
Fraunhofer Institute for
Experimental Software Engineering
Sauerwiesen 6
67661 Kaiserslautern, Germany
Phone: +49-6301-707-139
E-mail: muench@iese.fhg.de

Ove Armbrust
Fraunhofer Institute for
Experimental Software Engineering
Sauerwiesen 6
67661 Kaiserslautern, Germany
Phone: +49-6301-707-259
E-mail: armbrust@iese.fhg.de

Abstract

Empirical knowledge from software engineering studies is an important source for the creation of accurate simulation models. This article describes the development of a simulation model using empirical knowledge gained from an experiment at the NASA/GSFC Software Engineering Laboratory and from two replications at the University of Kaiserslautern. Data and analysis results are used to identify influence dependencies between parameters, and to calibrate models. The goal of the model is the determination of the effects (i. e., defect detection efficiency) of a requirements inspection process under varying contexts. The purpose is to provide decision support for project managers and process engineers when planning or changing a development process. This article describes the systematic model development with a focus on the use of empirical knowledge. Additionally, limitations of the model, lessons learned, and research questions for future work are sketched. The model performed well in an initial validation run, with only little deviation from experimental values.

1. Introduction

Today, new and innovative software engineering techniques are being developed constantly. Introducing them into a productive environment, however, is risky, because most effects are largely unknown. Even more importantly, well-known techniques such as inspections are not understood well enough in different contexts. Empirical software engineering helps in determining these effects, but is rather expensive, because experiments always require the inclusion of human subjects (similar to experiments in social sciences). Thus, simulation is becoming increasingly popular in the software engineering community. Integrating empirical knowledge into simulation is an important means for the creation of accurate models that reflect real world behavior for specific contexts. Nevertheless, many existing models are based purely on hypothetic assumptions that are not proven in real practice.

This article presents an example for the development of a simulation model by using empirical knowledge, and describes experiences with the modeling process. The goal is to demonstrate how empirical knowledge can be effectively used for simulation modeling, and to highlight typical challenges, difficulties, and limitations. The article is structured as follows: Section 2 motivates experiments in software engineering and highlights benefits of simulation modeling. Section 3 describes the empirical basis for the example model. The model development itself is described in Section 4, and model calibration and an extension to better suit real-world needs is explained in Section 5. Section 6 describes a validation run with data from an independent experiment, and Section 7 illustrates simulation results. In Section 8, we describe limitations of our model, followed by lessons learned. Section 9 summarizes our findings and gives an outlook on future work.

2. Experiments and Simulation

The engineering approach for solving the main problems in software development (time, cost, quality) proposes well understood and systematic processes. Traditional engineering disciplines like construction or mechanical engineering demonstrated both the need for and the benefits of a systematic process when larger projects were to be completed successfully. The same is true for software projects: Based on empirical findings, the (in other areas) well-known engineering principles help immensely in achieving

time, cost and quality goals [24]. Experimenting is one way to gather empirical findings and understand the effects of processes [4, 5]. Unfortunately, one typical characteristic of software development processes, techniques and methods is that their effects differ in different contexts. As a consequence, whatever is proven by one experiment must not necessarily be true for other contexts [11]. Another problem are the costs of experiments. Introducing simulations can help to overcome these deficiencies. According to [18, 13], important benefits of simulation in software engineering are cost reduction by simulating software development processes and human behavior and by better selecting and focusing the scope of real experiments; better demonstration of the benefits of new methods in the context of industrial development environments; and practice-oriented learning with respect to project planning and management (see [21] for an example approach).

One way of creating a reasonable connection between software process simulations and reality is the integration of empirical data. In general, combining empirical and process simulation knowledge can be done in the following ways: 1) Empirical knowledge is used for the development and calibration of simulation models, 2) results from process simulations are used for planning, designing and analyzing real experiments, and 3) process simulation and real experiments are performed in parallel (i. e., online simulation).

This paper focuses on the use of empirical knowledge for the development and calibration of simulation models. Following Münch et al. [18], we apply the following two cases of using empirical knowledge for process simulation:
1) Empirical data from controlled laboratory experiments (trends and behavior of software development parameters) is used for developing simulation models. This data helps to identify patterns and relations between parameters.
2) We use empirical data to initialize the input parameters and calibrate the simulation model.
3) Replications of empirical studies are used to increase the significance of empirical results. Generally, using data from replicated studies in simulation models improves the empirical basis of the model and can lead to better calibrations.

3. Empirical Knowledge

The empirical knowledge used for developing the simulation model was mainly gathered from the experiments described in [3] and [10]. The object of these experiments is a requirements document inspection process consisting of an individual preparation phase and a phase where the individually detected defects are pooled together into nominal teams. The initial experiment used professional software developers from the NASA/GSFC[1] Software Engineering Laboratory (SEL). We used data from an inspection of a generic set of documents, comparing the usual NASA reading technique with the perspective-based reading technique (PBR). The experiment was conducted in two runs. After a pilot run, several changes to the documents and the experimental settings were made. This is why we only consider results from the second run of the original experiment here. The replications took place one year later at the University of Kaiserslautern. All subjects were undergraduate students. The requirements documents were identical to the ones used in the initial experiment. This time, PBR was compared to an ad-hoc approach. Although the experiments were conducted to compare different reading techniques, and data collection was driven by this purpose, we used data like the average individual defect detection rate or team defect detection rates for the development of the simulation model.

[1]National Aeronautics and Space Administration/Goddard Space Flight Center

4. Simulation Model Development

The development of the simulation model followed a method that is currently being developed [23]. First, we defined the goals that we pursued with the model. Then, a static process model was created that reflects the inspection process. We used this static model as the basis for our dynamic simulation model. After that, we captured dynamic behavior and interrelations through an influence diagram. During this phase, we iteratively integrated empirical data from the experiments. Finally, we built the dynamic simulation model using the simulation tool Extend and calibrated it with data from the experiments. A validation run completed the model development.

4.1. Goals

The goal was to develop an empirically-based simulation model that allows for determining the effects of PBR and ad hoc requirements inspections in specific contexts. The intended usage of the model is to give project managers and process engineers decision support when planning or changing a development process (this can be combined with other decision support techniques as, for example, described in [22]). Furthermore, the model should help to reduce risks in introducing or changing requirements inspection processes by better forecasting their impacts. Several simulation models for inspection processes exist (e. g., [19, 20]), which are mainly based on hypothetical assumptions or expert knowledge. The scientific focus of this work was to better understand the use of quantitative empirical knowledge for simulation model development. Therefore, the model abstracts from several details and highlights the use of empirical data and further results from empirical studies.

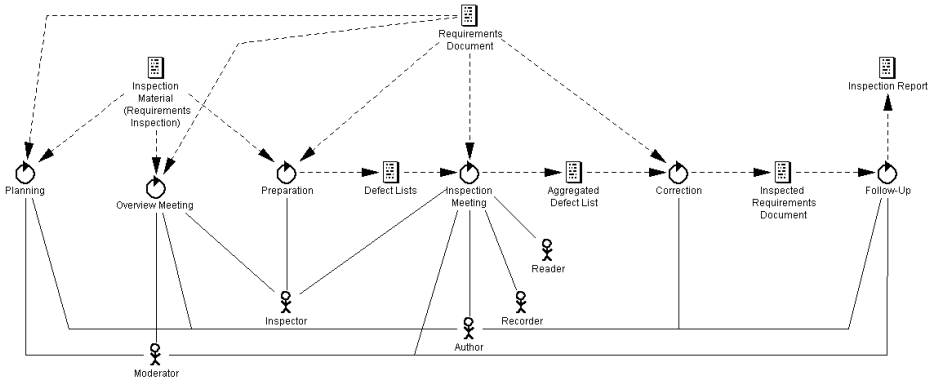

Figure 1. The inspection process model [17].

4.2. Analysis and Creation of the Static Process Model

Before modeling dynamic behavior, we developed a static process model that reflects the examined real-world process. This can be done, for instance, by descriptive software process modeling [6]. The static model of the inspection process we used is depicted in Figure 1. The activities, products and roles are modeled according to [15]. It is a well-known inspection process with individual preparation phases and a subsequent phase where individual defect lists are merged into one. Software engineering literature provides various experiments and studies where real team meetings are held and others where only nominal teams are examined. An overview can be found in [15]. There is no conclusive answer on whether team meetings enhance inspection results or not. The two replications compared real team meetings against nominal teams and found no increase in defect detection effectiveness in these settings. That is the reason why we also considered only nominal teams in our simulation model.

4.3. Creation of the Influence Diagram

An analysis of the data provided by the experiments revealed a relationship between individual defect detection rate (iDDR), team defect detection rate (DDR), and reviewer know-how. The defect detection rate is defined as percentage of the defects found with respect to all defects (Equation 1).

$$\textit{defect detection rate} = \frac{\textit{number of defects found}}{\textit{total number of defects}} \qquad (1)$$

iDDR indicates the individual performance of every reviewer, DDR specifies team performance with doubles sorted out. The experiments confirmed that DDR rises with iDDR, while iDDR was higher with professional software developers than with undergraduate students.

The description of the real experiments provided further information: the number of defects in the inspection documents was known (27/29) as well as the document size in pages (16/17). This results in an average of about 1.7 defects per page. Additionally, document complexity could be determined, e. g., using McCabe's cyclomatic complexity measure [16], since the documents are publicly available. We did, however, not include the number of defects as a determinant factor in the influence diagram or the model, because there was no empirical data available on how defect density affects inspections.

On the reviewer side, some attributes are also known. The original experiment used professional software developers as opposed to undergraduate students in the replication. While the number of years of experience did not correlate to inspection performance, the professionals yielded significantly higher average iDDR scores than the undergraduate students (28.39% versus 21.08%).

We chose to base our model on average defect detection ratios and to normalize other context information. We did this because the experiments also concentrate on average DDR scores. As can be seen in the influence diagram (Figure 2), iDDR values are derived directly or indirectly from the context variables. Thus, by focusing on defect detection ratios, we could decrease simulation complexity while at the same time still considering the context.

In Figure 2, we included literature references for positive or negative correlations in the influence diagram. References named as [Us] describe relations that we introduced to structure the model. This concerns primarily the intermediate attributes *Document Difficulty*, *EE Level* (i. e., experience and expertise level) and *Inspector Capability*. Basili et al. [3] found that the document reviewed significantly influences individual defect detection ratios in the generic problem domain. Biffl [7] found in his experiment that inspec-

tor's experience as well as expertise, as defined in Section 4.4, significantly influence inspection performance. Rising document difficulty lowers the inspector's defect detection ability for the specific document [3], while rising experience and expertise also raise inspector capability [7]: the better qualified an inspector is for the document to be inspected, the higher his individual defect detection rate can be expected to be.

We also needed to introduce an *iDDR Correction Factor* for the calculation of the team DDR. This factor symbolizes the overlap of iDDR values combined into a team. Depending on the reading technique, the defects found by each individual reviewer may overlap more or less due to restrictions resulting from the chosen reading technique. A PBR technique focusing on certain defect classes for each reviewer is more likely to yield a lower overlap than an ad-hoc technique, where every reviewer scans the complete document for any defects. For the immediate model purpose, the variables describing the context (i. e., inspector experience and document complexity) were not mandatory, since the context is already reflected in the iDDR values. Nevertheless, we included context variables in our model, but with neutral behavior. They are set to values that do not influence input iDDR values. To adapt the model to different contexts, the context variable values can be altered accordingly, with our context and its experiments as a reference value.

4.4. The Dynamic Model

As a first step towards modeling the influence diagram, we modeled variables that were used in the real experiments to capture real-world behavior (see Figure 2). Document difficulty, inspector capability, and EE Level are attributes defined by us in order to understand and describe dynamic behavior. They are not reflected in the final model as discrete blocks, but combined into some (more comprehensive) equations. This is purely a technical issue, their influence remaines the same. The modeling tool provides the possibility to create equations from discrete blocks such as *Add* or *Multiply*, or to enter the equation directly in an *Equation* block. This helps to reduce the number of blocks and connections in the model implementation significantly, thereby enhancing readability.

The following attributes are those we found to have a major influence on inspection results. We provided scales to measure these attributes, in order to alter their values and thereby adapt the model to new contexts. There are other scales that can be used, of course. In any case, it is possible to adapt the model to various concrete measures.

Document Attributes. Document attributes characterize the document that is to be inspected. Thus, they may differ with every document. The *Number of Defects (1..n)* influences the number of detected defects. In a defect-free document, no real defects can be detected, only false positives can be expected. A false positive is a reported defect in an actually correct part of the document. With an increasing number of defects, the number of actually detected defects can also rise. A natural limit can be expected at the point where the inspected document contains so many errors that no sensible defect detection can be done any more due to the fact that it gets difficult to identify defects at all. If this is the case, a document should be rejected, since it is not yet ready to be sensibly inspected. In our model, the number of defects is needed to visualize the effect of the inspection in terms of defect reduction. As mentioned earlier, the original experiments used documents with a defect density of about 1.7 defects per page. The *Document Size (0..2)* influences the inspection in at least two ways. First, the bigger the document gets, the more time is needed to read it and perform the defect detection. Second, with growing document size it becomes increasingly difficult to understand it completely, thus impeding defect detection [9]. Since in different domains very different document sizes occur, it does not seem sensible to use an absolute number here. Instead, a relative scale is used, with 1 symbolizing an average document size (average for the domain, company and review team). Numbers below 1 mean that the document is smaller than usual, while numbers above 1 mean an unusually large document. The value of 1 in our model represents our reference document with 17 pages, but document size can also be measured differently, of course. *Document Complexity (0..2)* also influences defect detection: A complex document may be very difficult to understand and therefore, finding any defects at all may be hard [9]—on the other hand, it may lead to more false positives than a very simple one. Thus, document complexity is a key attribute in the model. For quantifying this influence, the same considerations as for document size apply: In different domains, very different document complexities may appear. Therefore, the same relative scale as for document size was chosen. In order to use our model in different contexts, comparing document complexities, e. g., by using McCabe's cyclomatic complexity measure [16], seems plausible.

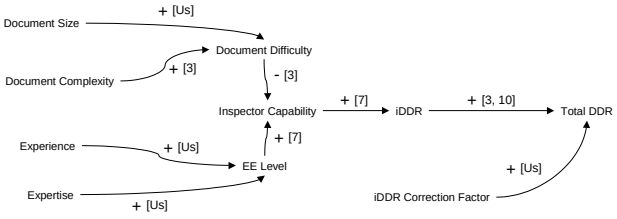

Figure 2. The influence diagram for the simulation model.

Reviewer Attributes. Reviewer attributes characterize the reviewers. Each reviewer has different capabilities, which must be taken into account. Reviewer attributes typically change only slowly over time, e. g., with growing domain knowledge. The model parameters, however, need to be adjusted for each different set of reviewers. *Experience (0..2)* symbolizes the reviewer's experience with inspections, the document type, and the reading technique chosen. We assume that this influences inspection performance: A reviewer using a new reading technique may need more time for administrative tasks than a more experienced reviewer, or an uncommon document type may lead to confusion, lowering the defect detection ratio. Finally, the reading technique also influences inspection effects. For simplicity, we measure experience on a relative scale with 1 symbolizing average experience (in the respective context), and values below and above symbolizing lower and higher experience. *Expertise (0..2)* takes into account the domain knowledge of the reviewer. While this might not be very important for some defect types, it may become absolutely necessary when looking at core system functions. When there are only reviewers foreign to the application domain, critical defects may go unnoticed until late in the development process. Thus, it is expected that domain knowledge also influences defect detection. Since the same scale problems as with experience apply to expertise as well, the same scale is used here. *Document Coverage (0..1)* symbolizes how much of the document can be covered by the reviewer in the allocated inspection time. Time pressure, for instance, can be expressed through this value. The values can be seen as percentage values. Normally, document coverage should be 1, but when time pressure inhibits a thorough inspection, the value can be lowered accordingly. *Individual Defect Detection Rate (0..1)* depicts the defect detection rate of every individual reviewer. Mostly, more than one reviewer will participate in an inspection. Thus, different individual defect detection rates must be considered. If the reviewer's capabilities are well-known, this value can be set accordingly. If only group performance has been monitored so far, mean and standard deviation values may be introduced at this point.

Additional Adjustment Variables. These additional input variables we deemed necessary to better describe the context of the inspection. When using a specific review technique, for example, experience with this specific technique may be more important than domain knowledge [7]. This is reflected in the simulation model through adjustment variables. If no special focus lies on one or more variables, all adjustment variables except for the review correction factor are set to 1. Lowering or raising a value increases or decreases the weighing of the corresponding factor relative to the other factors. This enables us to put special emphasis on single factors. *Experience Importance* describes the importance of reviewer experience. Experience may be especially important in special inspection processes with tailored reading techniques or special document types. *Expertise Importance.* If the software focuses on very special application domains such as aviation or weapons systems, domain expertise may become relatively more important than the other factors. This situation is reflected in this value. The *Review Correction Factor (0..1)* symbolizes the overlap of defect detection of the individual reviewers. Thus, it can change with the reading technique. An ad-hoc approach would have a high overlap since no special focus is given on where or how to detect defects, while a perspective-based approach would have a lower overlap. Altering this value could become necessary, for instance, if individual defect detection performance for a certain reading technique is known, but the search fields are redefined. If the new search fields supposedly overlap less, this value could be increased to reflect this context change in the model.

4.5. The Simulation Model

The model itself is realized as a discrete event model using the modeling and simulation environment Extend [14]. The document is generated with its properties, and a set of reviewers is generated with their properties. Then their individual defect detection rates are combined into the total defect detection rate of the (nominal) team. To illustrate the effects, the number of defects remaining in the document is calculated. An overview over the complete model is given in Figure 3.

The equation block labeled "iDDR" outputs the individual defect detection rate of the reviewer adjusted to the actual process context. It uses the attributes depicted in Figure 2 and the reviewer iDDR as input parameters. In our instantiation of the model, the block output equals input iDDR, because the context is already reflected in the input iDDR. The equation labeled "iDDR Correction" deals with the fact that not all defects detected by an individual are really new defects. Some have already been found by others, some not. Since the model is based on individual defect detection rates, these detection rates decrease with an increasing number of reviewers. This yields a potential problem if the group of reviewers is very inhomogeneous. If all reviewers perform about the same, it does not matter which one comes first and which one last. If there are major differences in defect detection ratio, their starting times in the model determines the simulation results. To eliminate this threat, several simulation runs with random starting numbers should be conducted and the results then averaged.

At a certain point, adding more reviewers does not lead to any more defects discovered. Although the number of reviewers needed to reach that point has not been provided by the real experiments, we included a constraint in the model

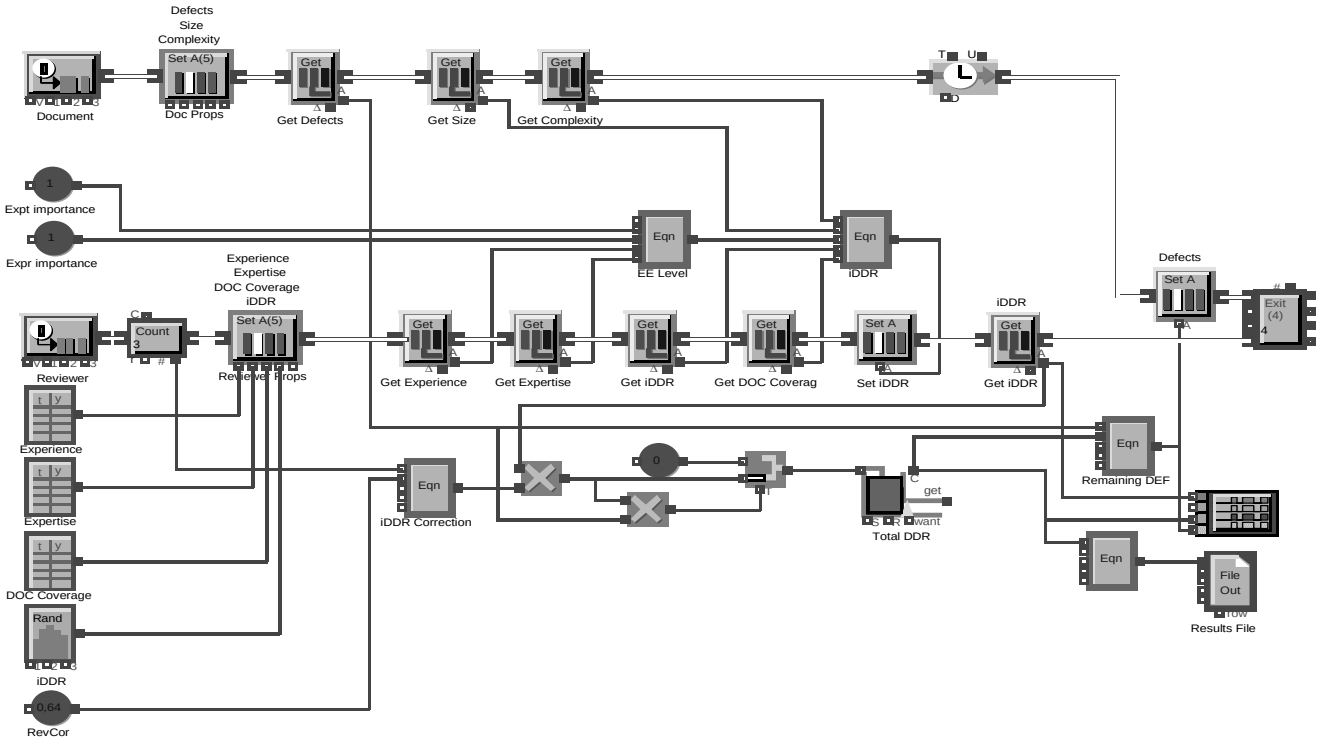

Figure 3. The Extend simulation model.

that discards individual defect detection success if the number of newly detected defects is smaller than one. After this check, the final adjusted defect detection rates are totaled up. For illustration purposes, the remaining defects are calculated last.

A simulation run consists of the following steps: At simulation time 0, the document item is created and its properties are set. In our case, defects were set to the number of defects in the reference document, while size and complexity were set to 1 in order not to spoil simulation results. The *Get* modules read out their respective values and provide them as input to the DDR calculation modules. Then the document item is halted for the time the reviewer items need to pass through the simulation.

At simulation time 1, the first reviewer item is created. The counter is needed for decreasing the number of newly detected defects for later reviewers. Again, the properties are set. Experience and expertise are set to 1, again not to spoil simulation results. We assumed a document coverage of 100%, so the corresponding value remains 1 throughout the simulation. Finally, iDDR gets set to values from a random number generator initialized with mean values from the reference experiments and a modest standard deviation of five percentage points of iDDR. This ensures that the simulation outputs a range of results in consecutive runs, so that mean results and typical variation can be identified. Experience and expertise importance were both set to 1, thus not preferring any of the two factors.

Following the setting of the reviewer properties, they are read again and used as inputs to various equations. EE Level adjusts the relative value of expertise and importance (Equation 2).

$$EE\ Level = \frac{Ec \cdot EcIm + Et \cdot EtIm}{EcIm + EtIm} \quad (2)$$

with

$$\begin{aligned} Ec &= \text{Experience} \\ EcIm &= \text{Experience Importance} \\ Et &= \text{Expertise} \\ EtIm &= \text{Expertise Importance} \end{aligned}$$

This result is combined with document size, complexity and coverage to an adjusted iDDR, which is then written back to the reviewer item (Equation 3), which then leaves the simulation. These equations represent the respective parts of the influence diagram.

$$adjusted\ iDDR = \frac{EELevel \cdot iDDR \cdot Coverage}{Size \cdot Complexity} \quad (3)$$

At simulation time 2, the next reviewer item enters the simulation and passes through the explained stages. This is re-

Table 1. Experiment Values.

Value	Pilot usual	Pilot PBR	1995 usual	1995 PBR	Rep95 ad-hoc	Rep95 PBR	Rep96 ad-hoc	Rep96 PBR
avg. DDR (%)	44,3	62,9	48,2	62,4	41,9	52,75	32,2	47,7
avg. iDDR (%)	20,58	24,92	24,64	32,14	21,33	25,93	14,59	22,46
avg. std. deviation (% DDR)	28,61	28,61	16,07	16,07	32,07	32,07	28,5	28,5
avg. DDR - 2*iDDR (%)	3,14	13,06	-1,08	-1,88	-0,76	0,89	3,02	2,78

peated until all reviewer items have passed through the simulation. Finally, the document item is released, the remaining number of defects is written back into the document, and the item exits the simulation, which stops the simulation run. All results are written to a file to simplify multiple runs and the analysis of the results.

Simply adding up the iDDR values calculated in this manner would, of course, lead to extremely high team defect detection rates, which could easily exceed 100%. Therefore, we adjusted the individual defect detection rates for later reviewers. This adjustment is calculated in the *iDDR Correction* equation. Determining this equation proved to be the hardest part of the modeling process. The original experiments did not provide detailed data about individual defect detection overlap. The only available information said that about 60% of the defects were found by more than one reviewer when using PBR, with a tendency to higher values when using ad-hoc techniques. This is not surprising, since PBR techniques try to overlap search fields only partially. While this did not give clear advice on how to decrease iDDR ratios for consecutive reviewers, it at least provided a general direction for defining a correctional factor. We determined the equation finally used by using data from the real experiments.

5. Model Calibration and Extension

In order to determine model behavior and calibrate the model, we integrated empirical data from the experiments described in [3] and [10]. This is explained in the following. We integrated empirical data from experiments not designed to support simulation models, but to compare different reading techniques. Therefore, we will dwell on points of special interest. Additionally, we introduce an extension to simplify real-world usage of the model.

5.1. Calibration

The two main data sources were the average individual defect detection rate (iDDR) and the average nominal team defect detection rate (DDR). Standard deviations extracted from the original experimental data were up to twice as high as the absolute iDDR values measured. Using these standard deviations in our model created reviewer iDDR values that were too unstable for reasonably calibrating the model. Therefore, we used a standard deviation of five percentage points of iDDR for calibration. In a real-world application of the model, however, realistic values should be used whenever available, since the standard deviation directly influences lowest and highest iDDR values. The values gathered in the real experiments are displayed in Table 1. We only integrated data from the 1995 run of the initial experiment and the two replications into the model. After the pilot run of the initial experiment, the experimenters changed the context of the experiment. Because of this, the results of the pilot and the other runs cannot be compared directly and, therefore, cannot be used for model calibration. Nevertheless, we had three sets of empirical data to calibrate our simulation model. Throughout the rest of this paper, we refer to these when we speak of experimental data. When examining the data, we noticed that no matter which reading technique was used, the average DDR was always about twice the individual DDR (Table 1). The highest deviation was 3.02 percentage points, with an average of 1.74. We do not believe this to be coincidence, so we used this fact for model calibration.

We did not model isolated defects, but relied on average values for defect detection. We did so because it reduced model complexity and because there was no complete data available from all three experiment runs concerning individual defects. Relying on average values made it more difficult to determine overlap in defect detection, though. By overlap, we mean defects that were detected by more than one reviewer. They increase individual DDR, but not team DDR. When not examining single defects, there is no possibility to determine how many of the defects detected by one team member had already been detected by others. We had, however, general information about how many of the total defects detected were found by how many persons. This gave us a clue as to where to look for a correctional factor.

Table 2. Validation Results.

Simulation dDDR	Remaining Defects	Experiment dDDR	Reality - Simulation
26,24%	63	27,0%	0,76%
24,11%	65	25,6%	1,49%
32,96%	58	33,5%	0,54%
24,11%	65	25,7%	1,59%

We then determined a value for our data sets through systematic simulation runs with different correctional factors.

We used an exponential decrease in the defect detection ratio. In our scenario, the first reviewer's iDDR is added fully to the total DDR, while the following iDDRs are decreased by a factor determined by the number of the respective reviewer and a review correction factor (RevCor). The iDDR that is added to the total DDR is calculated according to Equation 4.

$$\textit{final iDDR} \quad = \quad \textit{iDDR} \cdot \textit{RevCor}^{\textit{Count}} \qquad (4)$$

where Count is the number of the respective reviewer minus one. Systematically testing the model with different RevCor values, a value around 0.64 showed the lowest average deviation between the real experiments and the simulation results (see Figure 4). This can only be a first calibration, of course. More data from other experiments is needed to determine the validity of the equations we used and the RevCor factor.

5.2. Extension

After the initial model was developed and calibrated, we extended it to comprise a less abstract inspection process: Usually, more than one document is inspected, and several reviewer groups are involved. To cope with this, the extended version of the model was developed. Logically, there is no difference between the two versions. The extended version completely contains the initial version in that it uses the same document and reviewer flow and calculates the values of all variables similarly. All changes to the model are purely a technical matter, so we do not describe the extended model in detail here. For further information on the extended model, please refer to [2].

6. Model Validation

The model was developed to reflect a part of the real world, so a test of the model was needed. A good test data set would have a slightly changed context compared to the calibration data. [8] provides such a data set. The experiment originally analyzed the influence of team size and defect detection technique on the inspection effectiveness of a nominal team. Valid data from 169 undergraduate students with some software engineering experience was collected. The experimenters distinguished subjects using a checklist-based approach (denoted with "C" in the following), a scenario-based approach using the user viewpoint ("A"), the designer viewpoint ("D") and the tester viewpoint ("T"). All experiment runs were preceded by appropriate trainings.

Since the focus of the experiment was on reading technique mixes and team size, iDDR and team DDR data was only available for two-person teams. This creates a good test for the calculation of the document DDR value. The knowledge of the test subjects should be similar to the subjects from the replications used for calibration, since all were undergraduate students. Both cases looked at a requirements inspection in a classroom setting. The document was about twice as large (35 compared to 16/17) as the calibration documents and contained about three times as many defects (86 compared to 27/29). This makes for an average of 2.46 defects per page, which is 45 % higher than the calibration documents (1.7). So, there are some slight context changes.

For testing purposes, only combinations of two reviewers using the same reading technique were entered into the model. This should prevent spoiling simulation results caused by the combination of different reading techniques. Translated into model parameters, this means that four equal documents are inspected by two reviewers each: First AA (two "A" reviewers), then TT, CC, and finally DD. All other parameters kept their original values, including RevCor.

7. Simulation Results

The validation simulation run resulted in document DDR values very close to the experiment values. Table 2 gives an overview. The rows contain the values for the AA, TT, CC, and DD reviewer groups. The average deviation between simulation results and reality is 1.1 percentage points of DDR, which is very good.

The overall simulation results lead to the conclusion that the model does represent reality to a certain degree. It has become clear that integrating empirical data into models is a feasible way for obtaining good simulation models. It took about three person weeks to develop the simulation model, including the initial calibration and the extensions for the multi-document, multi-reviewer capabilities, with about one extra week for tool familiarization. An ar-

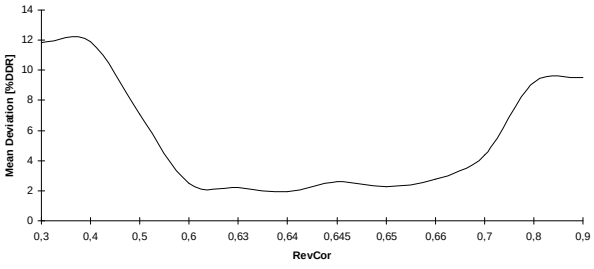

Figure 4. Deviation between experiments and simulation.

bitrary amount of time can be spent on further calibration with different data, of course. Up to now, no information is available on whether the model will pay off, and how many real experiments need to be replaced by simulations until break-even. This is also closely related to the reliability of the simulation results.

8. Discussion

Model Limitations. Our simulation model has several limitations. It describes a very abstract inspection process and includes only few factors. A standard inspection process is influenced by far more factors [1], most of which are not taken into account in this model. Our model relies on average defect detection ratios only. While this works fine for homogeneous groups of reviewers, problems arise when reviewers with very different iDDRs are combined. With our approach for adjusting each reviewer's iDDR, the result may depend on the time each reviewer starts. This threat should be met by repeatedly running the simulation with randomized starting times of the reviewers.

A third limitation concerns the *iDDR Correction* equation. It is based on data from three experiments, testing and intuition. It needs to be evaluated in other contexts, with more and less reviewers, and with different iDDR values. It served our purposes well in this model, but may be altered or replaced for other experiments. The influence of the adjustment variables influence is largely unknown as well. Currently, all factors are included in the equations in a linear manner. This raises questions like "With iDDR data available for a certain document size, will iDDR decrease in a linear way with increasing document size?" These questions will need to be answered in order to allow a reasonable and reliable scale-up.

Finally, the model was calibrated using data from three (similar) experiments only. Although a first validation run seems to confirm our setup, all equations, correction factors, and assumptions of our model need to be cross-checked in order to come to a more generally usable simulation model.

Lessons learned. We have learned several important things with respect to using empirical knowledge during the creation of this model. A lot of information that would have been helpful for creating the simulation model was not available. This was not because it is difficult to gather it, but because we used data from experiments not designed to fit the needs of a simulation model. With only little more effort, the desired data could have been collected, which would have simplified model building a great deal. This might be an issue for the design of future experiments.

Furthermore, deriving valid equations from discrete data points is also not an easy task. Especially the combination of the RevCor factor and the iDDR Correction equation proved to be difficult. Unfortunately, the experiments themselves could not provide much helpful information. This was not because of the experimental setup, but because of the lack of determined interrelations between influence factors and the amount of different contexts. We tried to overcome this by extensively testing out ideas, but we failed to provide a more systematic approach.

The empirical knowledge we used mainly resulted from one experiment and two replications. Although this was considered in the design of the replications, the contexts were not completely identical. Additionally, we used empirical knowledge from other empirical studies with different contexts for identifying further impact dependencies. It seems to be a challenging research field to investigate to which extent context variations are acceptable and what the consequences for the external validity of the model are (further details about threats to validity can be found in [12]). Guidance for this topic is missing. From our point of view, a careful commonality analysis of the contexts could be a starting point. Finally, using a visual modeling tool such as the one we used proved to be a great help. Visualizing the model modules and their connections allowed for effectively developing the model. A text- or source code-based model would have been significantly harder to develop.

9. Summary and Outlook

In this paper, we presented a discrete-event simulation model of an inspection process. The model is based on empirical data from an inspection experiment at the NASA/GSFC Software Engineering Laboratory and two replications at Kaiserslautern University. The experiments provided data about average individual defect detection rates of the reviewers and average total team defect detection rates. The simulation model reconstructs the inspection process in an abstract form. Several adjustment variables allow for future adaptation of the model to different contexts. The model itself is based on average defect detection ratios.

We encountered several problems during modeling and integration of empirical knowledge. The original experiments did not provide all the data we would have liked for our model. Also, deriving valid equations from discrete data points is not an easy task and requires extensive explorative studies. One possible direction for future research would be the iDDR correction/RevCor area. More data from different experiments is needed to determine the validity of our proposals. Another important direction would be to explore the possibilities of adapting the model to different contexts. Here, especially our integration of what we call the "adjustment variables" could be enhanced.

Finally, more methodological support for combining empirical data and simulation modeling is needed. This could lead to a new and advanced type of experimental laboratory that uses process simulation as a virtual capability.

Acknowledgments

This work was supported in part by the German Federal Ministry of Education and Research (SEV Project) and the Stiftung Rheinland-Pfalz für Innovation (ProSim Project, no.: 559). We would like to thank Sonnhild Namingha from the Fraunhofer Institute for Experimental Software Engineering for reviewing the first version of the article.

References

[1] O. Armbrust. Developing a characterization scheme for inspection experiments. Project Thesis, University of Kaiserslautern, Germany, 2002.

[2] O. Armbrust. Using empirical knowledge for software process simulation: A practical example. Diploma Thesis, University of Kaiserslautern, Germany, May 2003.

[3] V. R. Basili, S. Green, O. Laitenberger, F. Lanubile, F. Shull, S. Sørumgård, and M. V. Zelkowitz. The empirical investigation of perspective-based reading. *Empirical Software Engineering*, 1(2):133–164, October 1996.

[4] V. R. Basili, F. McGarry, R. Pajerski, R., and M. Zelkowitz. Lessons learned from 25 years of process improvement: The rise and fall of the NASA Software Engineering Laboratory. In *Proceedings of 24th International Conference on Software Engineering (ICSE), Orlando, Florida, USA*, May 2002.

[5] V. R. Basili, R. Selby, and D. Hutchens. Experimentation in software engineering. *IEEE Transactions on Software Engineering*, July 1986.

[6] 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), Kaiserslautern, Germany*, 2188, pages 312–325. Lecture Notes in Computer Science, September 2001.

[7] S. Biffl. Analysis of the impact of reading technique and inspector capability on individual inspection performance. In *Proceedings of the 7th Asia Pacific Software Engineering Conference (APSEC), Singapore*, December 2000.

[8] S. Biffl and W. Gutjahr. Analyzing the influence of team size and defect detection technique on the inspection effectiveness of a nominal team. In *Proceedings of the 7th International Software Metric Symposium (METRICS), London, England*, April 2001.

[9] L. C. Briand, O. Laitenberger, and I. Wieczorek. Building resource and quality management models for software inspections. Technical Report 97-06, Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern, Germany, 1997.

[10] M. Ciolkowski, C. Differding, O. Laitenberger, and J. Münch. Empirical investigation of perspective-based reading: A replicated experiment. Technical Report 048.97/E, Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern, Germany, December 1997.

[11] M. Joseph. Software engineering: Theory, experiment, practice or performance. Technical report, University of Warwick, England, 1988.

[12] C. Judd, E. R. Smith, and L. Kidder. *Research Methods in Social Relations*. Harcourt Brace Jovanovich College Publishers, 6 edition, 1986.

[13] M. I. Kellner, R. J. Madachy, and D. M. Raffo. Software process simulation modeling: Why? What? How? *Journal of Systems and Software*, 46(2–3):91–105, 1999.

[14] D. Krahl. The Extend simulation environment. In J. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, editors, *Proceedings of the 2000 Winter Simulation Conference (Wintersim), Orlando, Florida, USA*, pages 280–289. IEEE Press, 2000.

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

[16] T. McCabe. A software complexity measure. *IEEE Transactions on Software Engineering*, 2(4):308–320, 1976.

[17] J. Münch, T. Berlage, T. Hanne, H. Neu, S. Nickel, S. von Stockum, and A. Wirsen. Simulation-based evaluation and improvement of software development processes. SEV Progress Report No. 1, Technical Report No. 048.02/E, Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern, Germany, 2002.

[18] J. Münch, D. Rombach, and I. Rus. Creating an advanced software engineering laboratory by combining empirical studies with process simulation. In *Proceedings of the International Workshop on Software Process Simulation and Modeling (ProSim), Portland, Oregon, USA*, May 2003.

[19] H. Neu, T. Hanne, J. Münch, S. Nickel, and A. Wirsen. Simulation-based risk reduction for planning inspections. In M. Oivo and S. Komi-Sirviö, editors, *Proceedings of the 4th International Conference on Product Focused Software Process Improvement (Profes), Rovaniemi, Finland*, pages 78–93. Lecture Notes in Computer Science 2559, December 2002.

[20] H. Neu, T. Hanne, J. Münch, S. Nickel, and A. Wirsen. Creating a code inspection model for simulation-based decision support. In *Proceedings of the International Workshop on Software Process Simulation and Modeling (ProSim), Portland, Oregon, USA*, May 2003.

[21] D. Pfahl. *An Integrated Approach to Simulation-Based Learning in Support of Strategic and Project Management in Software Organisations*. PhD thesis, Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern, Germany, 2001.

[22] I. Rus, S. Biffl, and M. Halling. Systematically combining process simulation and empirical data in support of decision analysis in software development. Technical report, Software Engineering and Knowledge Engineering (SEKE), 2002.

[23] I. Rus, H. Neu, and J. Münch. A systematic methodology for developing discrete event simulation models of software development processes. In *Proceedings of the International Workshop on Software Process Simulation and Modeling (ProSim), Portland, Oregon, USA*, May 2003.

[24] I. Sommerville. *Software Engineering*. Addison-Wesley, 1987.