Software Process Simulation

Chapter · March 2012

DOI: 10.1007/978-3-642-24291-5_7

7 Software Process Simulation

This chapter introduces software process simulation as a means to amend and complement empirical studies, for example, to evaluate changing contexts, and to analyze process dynamics. It introduces two types of simulation models, namely continuous and discrete-event models, as well as their combination in hybrid models. In addition, this chapter describes a systematic method for the creation of simulation models and introduces an existing library of simulation model components that can be easily reused. Finally, it explains how process simulation can be combined with empirical studies to accelerate process understanding and improvement. Fig. 7.1 displays the chapter structure.

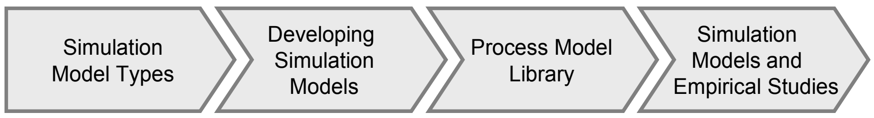

Simulation Model Types
Developing Simulation Models
Process Model Library
Simulation Models and Empirical Studies

Fig. 7.1: Chapter structure

7.1 *Objectives of this chapter*

After reading this chapter, you should be able to

- explain the role of simulation in software process engineering,
- name and explain the two major types of software process simulation,
- explain their combination into hybrid simulation, and
- understand the development of a simulation model.

7.2 Software process simulation

While experiments are a valuable tool for evaluating the effects of a specific process (change), it may not always be possible to perform such experiments. In this case, simulations may help.

Software process engineering has only recently started to discover the possibilities of simulation. In this section, the benefits of process simulations will be discussed, as well as the question of what to simulate. Two different simulation approaches will be introduced later in this chapter, as will their combination.

Three main improvement classes benefitting from software development process simulation can be identified: cost, time, and knowledge improvements. Cost improvements originate from the fact that conventional experiments are very costly. The people needed as experimental subjects are usually employees. This makes every experimentation hour expensive, since the subjects get paid while not immediately contributing to the company's earnings. Conducting a simulation instead of a real experiment saves the expenses for experimental subjects.

Time benefits can be expected from the fact that simulations can be run at (almost) any desired speed. While an experiment with a new project management technique may take months, the simulation may be sped up almost arbitrarily by simply having simulation time pass faster than real time. On the other hand, simulation time may be slowed down arbitrarily. This is done when simulating biological processes, for example. It might be useful in a software engineering context when too much data is accumulated within too little time and therefore cannot be analyzed properly. While in the real world, decisions would have to be based on partial information, the simulation can be stopped and the data analyzed. When the analysis is complete, the simulation can be continued with a decision based on the completely analyzed data.

Knowledge improvements stem mainly from two areas: Simulation can be used for training purposes and experiments can be replicated in different contexts. Training software engineers in project management, for example, requires a lot of time. The trainee needs to make mistakes to learn from, which in turn cost time and money in the project, and has to be instructed to prevent him from making really bad mistakes that would cost even more time and money. Training people in a laboratory setting might be even worse. Using a simulation environment such as the one introduced in [1] enables the trainee to experience immediate feedback to his decisions. The consequences of choosing a certain reading technique, for example, do not occur months after the decision, but minutes. In this way, the complex feedback loops can be better understood and mistakes can be made without endangering everyday business.

Simulations can also be used to replicate an experiment in a different context, for example with less experienced subjects. If the properties of the experimental objects are sufficiently explored, the consequences of such changes can be examined in simulations instead of costly experiment replications. Learning from simulations can save both time and money compared to real experiments.

Another useful application of simulation are high-risk process modifications. These may be (yet) uncommon processes, or catastrophe simulations. An example of the former is Extreme Programming, which seemed to contradict many software engineering principles of documentation and planning at first, but has proved to be beneficial in certain situations after all. When high-risk process changes are to be examined, simulations can provide a sandbox in which the changes can be tried out without any consequences. If the results show the proposed changes to be a complete letdown, at least no money was spent on expensive experiments.

A catastrophe simulation can investigate extreme process changes, for example the loss of key personnel in a project. While it is clear that this will influence the process and its outcome noticeably, determining quantitative effects can only be done in an experiment. Will a project be delayed by 20%, or by 200%? What about product quality? Since this situation rarely occurs, this kind of real experiment is not conducted because of the high costs associated with it. A simulation does not have such high costs. It probably also does not forecast the exact numbers, but nevertheless, it shows their magnitude and, as its main benefit, helps to better understand the situation and problem/challenge at hand. It may also show key problem areas, which may then be addressed in real life in order to absorb the worst consequences of the hypothetical catastrophe.

The following classification of simulations follows Kellner et al. [2]. Kellner et al. have determined relationships between a model's purpose and what has to be modeled, and have classified approaches on how to simulate.

When a simulation is to be set up, the first question to be answered is about the *purpose.* What is the simulation to be used for? Is it for strategic planning, or for training, or for operational process improvement? After having answered this question, it is possible to determine *what to include* in the simulation, and *what to leave out.* To structure this decision, the *model scope, result variables, process abstraction,* and *input parameters* can be distinguished.

The *model scope* must be adapted to the model purpose. Let's say a software company wants to test a new reading technique for defect detection. Clearly, the inspection process must be modeled, but changes to the reading technique will most likely have other effects later in the development process. The defect density might be higher, therefore lowering product quality and increasing rework time. This must be considered in the model because otherwise, the impact of the process change will not be reflected correctly in the model.

According to Kellner et al. [2], the model scope usually ranges from a part of the lifecycle of a single product to long-term organizational considerations. They introduced two sub-categories: *time span* and *organizational breadth*. *Time span* is proposed to be divided into three sub-categories: less than 12 months, 12–24 months, and more than 24 months. *Organizational breadth* considers the number of product/project teams: less than one, exactly one, or multiple teams involved.

The *result variables* are mandatory for answering the questions posed when determining the model purpose. Variables can be thought of as information sources in an abstract sense here; however, most models include variables such as costs, effort, time consumption, staffing needs, or throughputs. The choice of variables once again depends on the purpose of the model. If the purpose is to predict overall end-of-project effort, different variables must be measured than the ones needed for predictions at the end of every major development step.

Questions like these also influence the *level of abstraction* at which the model is settled. If effort at every major development step is to be determined, the model probably needs finer granularity than if only overall end-of-project effort is the focus. In any case, it is important to identify key activities and objects (documents) as well as their relationships and feedback loops. In addition, other resources such as staff and hardware must be considered. Depending on the level of abstraction, this list will get more or less detailed.

Finally, *input parameters* must be determined. They depend largely on the desired output variables and the process abstraction. In general, many parameters are needed for software process simulation; the model by Abdel-Hamid and Madnick [3] requires several hundreds. Kellner et al. [2] provide some examples: effort for design and code rework, defect detection efficiency, personnel capabilities, etc. Fig. 7.2 illustrates the relationships among the aspects described above according to [2].

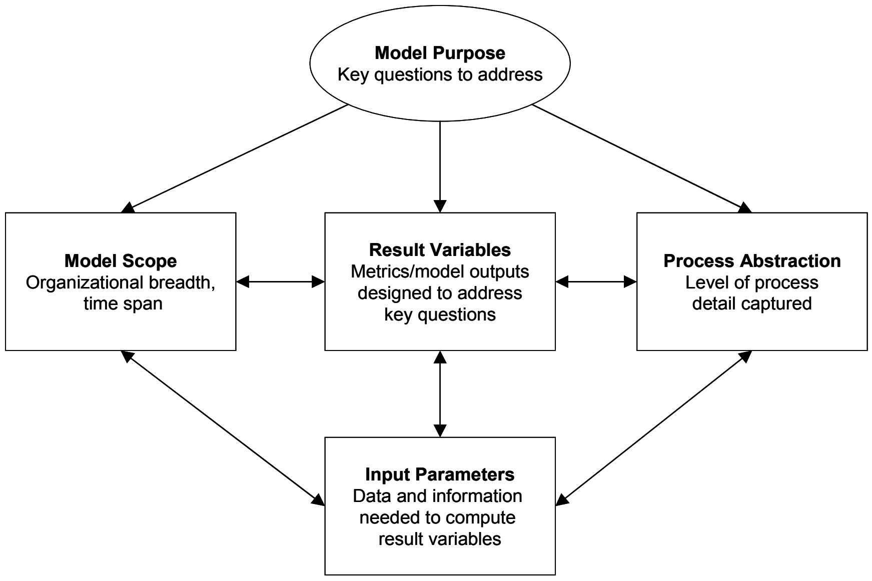

Model Purpose
Key questions to address
Model Scope
Organizational breadth, time span
Result Variables
Metrics/model outputs designed to address key questions
Process Abstraction
Level of process detail captured
Input Parameters
Data and information needed to compute result variables

Fig. 7.2: Relationships among model aspects

After determining the purpose of the model and what to include, the *choice of a simulation technique* is next. The continuous and the discrete simulation approaches will be introduced in Sections 7.2.1 and 7.2.2, as will their combination (Section 7.2.3). In addition to that, there are several state- and rule-based approaches as well as queuing models [4].

Simulation techniques may be distinguished into *visual* and *textual models.* Most models support some kind of visual modeling today in order to facilitate development of the models. Some modeling tools support a semi-automatic validation functionality, e.g., Fraunhofer IESE's Spearmint [5]. Appropriate possibilities for specifying interrelationships among entities of the model enable accurate modeling of the real world. Continuous output of result variables as opposed to only presenting the results allows monitoring the simulation run and providing early intervention. Interactive simulations allow for altering input variables during execution.

Simulations can be entirely deterministic, entirely stochastic, or a mix of both. Entirely deterministic models use input parameters as single values without variation. The simulation needs to be run only once to determine its results. A stochastic simulation assumes random numbers out of a given probability distribution as input parameters. This recognizes the inherent uncertainty in software development, especially when evaluating human performance. Consequentially, several runs are needed to achieve a stable result. Batch runs of simulations with changed input parameters simplify this approach significantly.

A sensitivity analysis explores the effects of input variable variations on the result variables. This has two advantages: First, the researcher knows how much variation in the results has to be expected due to variations in input parameters, and second, he/she can identify the parameters with the biggest influence on the result. These should be handled and examined with special care.

Finally, to obtain valid results from the simulation, calibrating the simulation model against the real world is necessary. This should be done by integrating actually measured data into the model, and comparing the results of simulations of real processes with the respective real-world values. Accurate measuring of input parameters and result variables is mandatory here. In many cases, however, no suitable real-world data is available. In this case, Kellner et al. [2] suggest trying to construct the data needed from available data (e.g., constructing cost data from effort data) or retrieving it from original documents, rather than final reports, or obtaining estimates from the staff or from published values.

7.2.1 Continuous simulation

In 1972, the Club of Rome started an initiative to study the future of human activities on our planet. The initiative focused on five physical and easily measurable quantities: population, food production, industrial capital, production, and non-renewable natural resources [6]. A research group at the Massachusetts Institute of Technology (MIT) developed a societal model of the world. This was the first continuous simulation model: the World Model [7].

Today, most continuous models are based on differential equations and/or iterations, which use several input variables for calculation and in turn supply output variables. The model itself consists of nodes connected through variables. The nodes may be instantaneous or non-instantaneous functions. Instantaneous functions present their output at the same time the input is available. Non-instantaneous functions, also called memory functions, take some time for their output to change.

This approach of a network of functions allows simulating real processes continuously. An analogy would be the construction of mathematical functions (add, subtract, multiply, integrate) with analog components like resistors, spools, or condensers. Even complex functions containing partial differential equations that would be difficult or impossible to solve analytically or numerically may be modeled using the three basic components mentioned above. Before computers became as powerful as they are today, the analog approach was the only way to solve this kind of equations within a reasonable amount of time. Due to the continuous nature of the “solver”, the result could be measured instantly.

Of course, simulating this continuous system on a computer is not possible due to the digital technology used. To cope with this, the state of the system is computed at very short intervals, thereby forming a sufficiently correct illusion of continuity.

This iterative re-calculating makes continuous models simulated on digital systems grow complex very fast.

In software engineering contexts, continuous simulation is used primarily for large-scale views of processes, like the management of a complete development project, or strategic company management. Dynamic modeling enables us to model feedback loops, which are very numerous and complex in software projects. The well-known System Dynamics framework for continuous simulation models is described in great detail in [8].

7.2.2 Discrete-event simulation

The discrete approach shows parallels to clocked operations like those used by car manufacturers in production. The basic assumption is that the modeled system changes its state only at discrete moments of time, as opposed to the continuous model. So, every discrete state of the model is characterized by a vector containing all variables, and each step corresponds to a change in the vector.

Example. Let's consider a production line at a car manufacturer. Simplified, there are parts going in on one side and cars coming out on the other side. The production itself is clocked: Each work unit has to be completed within a certain amount of time. When that time is over, the car-to-be is moved to the next position, where another work unit is applied. (In reality, the work objects move constantly at a very low speed. This is done for commodity reasons and to realize minimal time buffer. Logically, it is a clocked sequence.) This way, the car moves through the complete factory in discrete steps.

Simulating this behavior is easy with the discrete approach. Each time a move is completed, a snapshot is taken of all production units. In this snapshot, the state of all work units and products (cars) is recorded. At the next snapshot, all cars have moved to the next position. The real time that passes between two snapshots or simulation steps can be arbitrary. Usually the next snapshot of all variables is calculated and then the simulation assigns the respective values. Since the time needed in the factory to complete a production step is known, the model appropriately describes reality.

A finer time grid is certainly possible: Instead of viewing every clock step as one simulation step, arrival at and departure from a work position can be used, thereby capturing work and transport time independently.

The discrete approach is used in software engineering as well. One important area is experimental software engineering, e.g., regarding inspection processes. Here, a discrete simulation can be used to describe the process flow. Possible simulation steps might be the start and completion of activities and lags, together with special events like (late) design changes. This enables discrete models to represent queues.

7.2.3 Hybrid simulation

Continuous simulation models describe the interaction between project factors well, but suffer from a lack of detail when it comes to representing discrete system steps. Discrete simulation models perform well in the case of discrete system steps, but make it difficult to describe feedback loops in the system.

To overcome the disadvantages of the two approaches, a hybrid approach as described by Martin and Raffo [9] can be used. In a software development project, information about available and used manpower is very important. While a continuous model can show how manpower usage levels vary over time, a discrete model can point out bottlenecks, such as inspections. Because of insufficient manpower, documents are not inspected near-time, so consumers of those documents have to wait idly, which wastes manpower.

In a purely discrete approach, the number of inspection steps might be decreased to speed up the inspection process. While possibly eliminating the bottleneck, this might introduce more defects. The discrete model would not notice this until late in the project because continually changing numbers are not supported. The hybrid approach, however, would instantly notice the increase, and – depending on how the model is used for steering the project – more time would be allocated for rework, possibly to the extent that the savings in inspection are overcompensated. Thus, the hybrid approach helps in simulating the consequences of decisions more accurately than each of the two single approaches does individually.

7.2.4 Benefits

Following Kellner et al. [2], benefits in the following areas can be expected from simulation in software engineering:

– *Strategic management issues.* These may be questions such as whether to distribute work or concentrate it in one spot, or whether development should be done in-house or be outsourced, or whether to follow a product-line approach or not.

– *Planning.* Planning includes forecasting schedule, costs, product quality, and staffing needs, as well as considering resource constraints and risk analysis.

– *Control and operational management.* Operational management comprises project tracking, an overview of key project parameters such as project status and resource consumption, comparison of actual to planned values, as well as operational decisions such as whether to commence the next major phase (e.g., coding, integration testing).

– *Process improvement and technology adoption.* This includes evaluating and prioritizing suggested improvements before they are implemented as well as *ex post* comparisons of process changes against simulations of the unchanged process with actually observed data.

– *Understanding.* Simulation models can help process members to better understand process flow and the complex feedback loops usually found in software development processes. Also, properties pervading many processes can be identified.

– *Training and learning.* Similar to pilots training in flight simulators, trainees can learn project management with the help of simulations. The consequences of mistakes can be explored in a safe environment and experience can be collected that is equivalent to years of real-world experience.

7.3 A method for developing simulation models

This section summarizes a method for systematically developing discrete-event software process simulation models, which was previously published in [10]. However, even though used for developing a discrete-event model, it can also be used to develop continuous simulation models. The method considers the development of a new simulation model without reusing or incorporating existing components. If reuse is considered (by either incorporating existing components or developing for reuse), the method has to be changed to address possible reuse of simulation model elements.

The lifecycle of a simulation model for long-term use is similar to that of software, and consists of three main phases: *development, deployment,* and *operation,* including *maintenance and evolution.* The activities within these phases can have different temporal orders and dependencies; therefore, the resulting lifecycle can take on different forms, such as waterfall, iterative, or even agile.

The activities throughout the lifecycle can be divided into two categories: *engineering* and *management* activities.

The engineering activities for model development are (Fig. 7.3):

– requirements identification and specification for the model to be built,

– analysis and specification of the modeled process,

– design of the model,

– implementation of the model, and

– verification and validation throughout development.

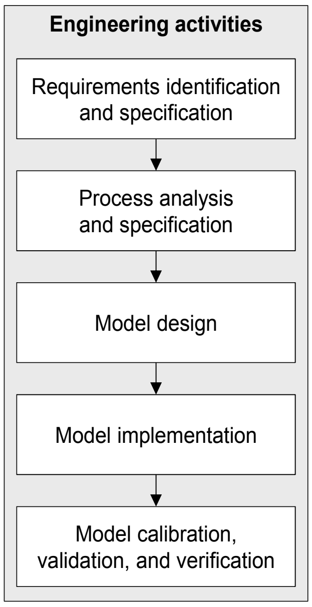

Engineering activities
Requirements identification and specification
Process analysis and specification
Model design
Model implementation
Model calibration, validation, and verification

Fig. 7.3: Simulation model development engineering activities

The management activities are (Fig. 7.4):

- model development planning and tracking,
- measurement of the model and of the model development process, and
- risk management (this refers to identifying and tracking risk factors, and mitigating their effects. Some of the risk factors are: changes in customer requirements, changes in the description of the modeled process, non-availability of data for the quantitative part of the model.)

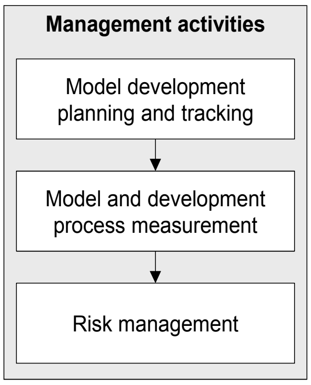

Management activities
Model development planning and tracking
Model and development process measurement
Risk management

Fig. 7.4: Simulation model development management activities

During the lifecycle of a simulation model, different roles are involved. In the development phase, mainly the model developer and the customer are involved. A "pair modeling" approach for creating the first version of the static process model and influence diagram can be very useful during this phase, because the discussion about the model and the influences is very inspiring. The following sections introduce the engineering activities in more detail.

7.3.1 Requirements identification and specification

During the requirements activity, the purpose and the usage of the model have to be defined. According to this, the questions that the model will have to answer are determined and so is the data that will be needed in order to answer these questions. The steps of the requirements specification are:

7.3.1.1 Definition of the goals, questions, and the necessary metrics

A goal-question-metrics-based approach like GQM can be used for defining the goal and the needed measures [11]. GQM can also be used to define and start an initial measurement program if needed. Purpose, scope, and level of detail for the model are described by the *goal*. The *questions* that the model should help to answer are formulated next. Afterwards, parameters (*metrics*) of the model (outputs) have to be defined that (once their value is known) will answer the questions. Then those model input parameters have to be defined that are necessary for determining the output values. The input parameters should not be considered as final after the requirements phase; during the analysis phase, they will usually change.

7.3.1.2 Definition of usage scenarios

Define scenarios ("use cases") for using the model. For example, for answering the question: "How does the effectiveness of inspections affect the cost and schedule of the project?", a corresponding scenario would be: "All input parameters are kept constant and the parameter *inspection effectiveness* is given x different values between (min, max). The simulator is executed until a certain value for the number of defects per KLOC is achieved, and the values for *cost* and *duration* are examined for each of the cases." For traceability purposes, scenarios should be tracked to the questions they answer (for example by using a matrix).

7.3.1.3 Development of test cases

Test cases can be developed in the requirements phase. They help to verify and validate the model and the resulting simulation.

7.3.1.4 Validation of requirements

The customer (i.e., the organization that is going to use the simulation model) has to be involved in this activity and must agree with the content of the resulting model specification document. Changes can be made, but they have to be documented.

Throughout this section, we will illustrate the outputs of the activities by using some excerpts from a discrete-event simulator. This model and simulator support managerial decision-making for planning the system testing phase of software development. The simulator offers the possibility of executing *what-if* scenarios with different values for the parameters that characterize the system testing process and the specific project. By analyzing the outputs of the simulator, its user can visualize predictions of the effects of his/her planning decisions.

7.3.1.5 Examples for step 1 and step 2:

Goal

- Develop a decision support model for the planning of the system testing phase in the context of organization x such that trade-offs between duration, cost, and quality (remaining defects) can be analyzed and the most suitable process planning alternative can be selected.

Questions to be answered

- Q1: When to stop testing in order to achieve a specified quality (number of defects expected to remain in the delivered software)?
- Q2: If the delivery date is fixed, what will be the quality of the product if delivered at that time, and what will be the cost of the project?
- Q3: If the cost is fixed, when will the product have to be delivered and what will be its quality at that point?
- Q4: Should regression testing be performed? To what extent?

Output parameters

- O1: Cost of testing (computed from the effort) [$] for cost or [staff hours] for effort
- O2: Duration of testing [hours] or [days]
- O3: Quality of delivered software [number of defects per K lines of code]

(Some of the) Input parameters:

- I1: Number of requirements to be tested
- I2: Indicator of the "size" of each software requirement (in terms of software modules (components) that implement that requirement and their "complexity/difficulty" factor)
- I3: For each software module, its "complexity/difficulty" factor
- I4: Number of resources (test-beds and people) needed

- I5: Number of resources (test-beds and people) available
- I6: Effectiveness of test cases (historic parameter that gives an indication of how many defects are expected to be discovered by a test case)

Scenarios

- S1: For a chosen value of the desired quality parameter, and for the fixed values of the other inputs, the simulator is run once until it stops (i.e., the desired quality is achieved) and the values of the cost and duration outputs are examined.
- S2: The simulator is run for a simulation duration corresponding to the chosen value of the target duration parameter, and for the fixed values of the other inputs. The values of the cost and quality outputs are examined.
- S3: For a chosen value of the desired cost parameter, and for the fixed values of the other inputs, the simulator is run once until it stops (i.e., the cost limit is reached) and the values of the quality and duration outputs are examined.
- S4: For a chosen value of the desired quality parameter, and for the fixed values of the other inputs, the simulator is run once until it stops (i.e., the desired quality is achieved) and the values of the cost and duration outputs are examined according to the variation in the extent of regression testing.
- S5: The simulator is run for a simulation duration corresponding to the chosen value of the target duration parameter, and for the fixed values of the other inputs, and the values of the cost and quality outputs are examined according to the variation in the extent of regression testing.
- S6: For a chosen value of the desired cost parameter, and for the fixed values of the other inputs, the simulator is run once until it stops (i.e., the cost limit is reached) and the values of the quality and duration outputs are examined according to the variation in the extent of regression testing.

7.3.2 Process analysis and specification

The understanding, specification, and analysis of the process that is to be modeled is one of the most important activities during the development of a simulation model.

Process analysis and specification can be divided into four sub-activities, as shown in Fig. 7.5:

- analysis and creation of a static process model (“a”, straight line),
- creation of the influence diagram for describing the relationships between parameters of the process (“b”, dashed line),
- collection and analysis of empirical data for deriving the quantitative relationships (“c”, dash-dot line), and
- quantification of the relationships (“d”, dash-dot-dot line).

Fig. 7.5 sketches the product flow of this activity, i.e., it describes which artifacts (document symbol) are used or created by each task (arrowed circle symbol).

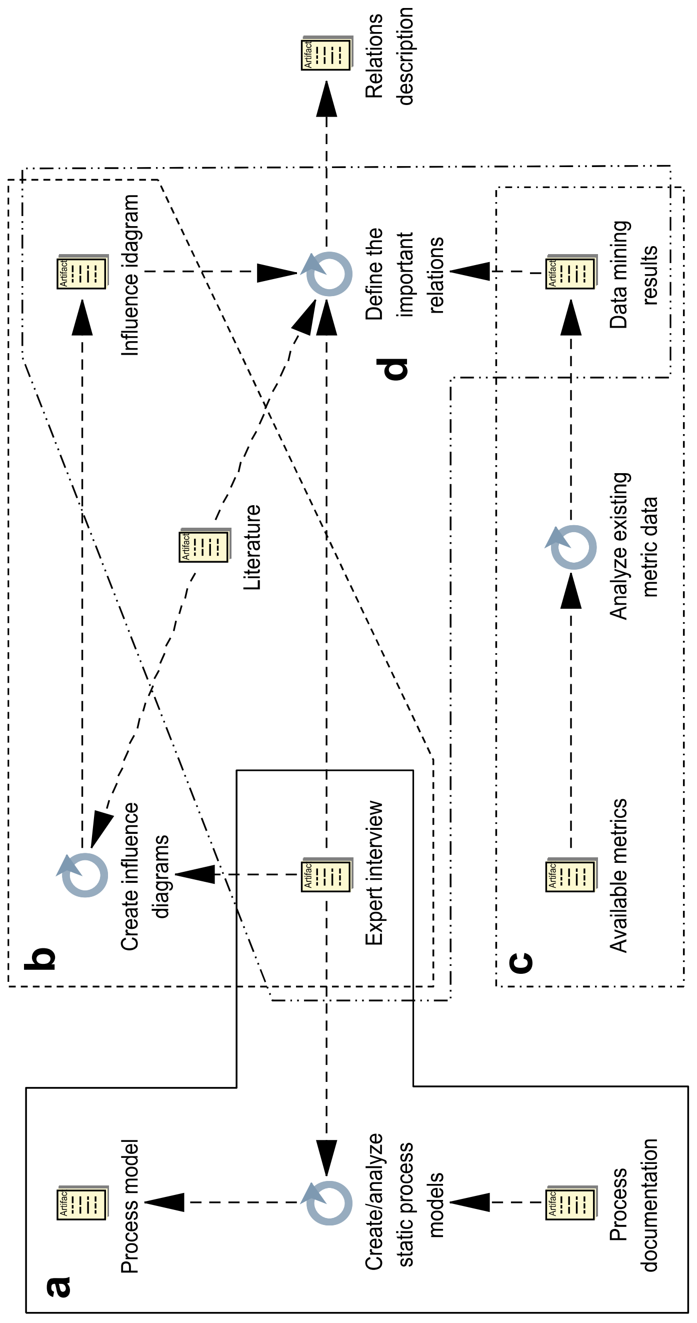


Fig. 7.5: Process analysis and specification

a: Analysis and creation of a static process model

The software process to be modeled first needs to be understood and documented. This requires that the representations (abstractions) of the process should be intuitive enough to be understood by the customer and to constitute a communication vehicle between modeler and customer. These representations lead to a common definition and understanding of the object of modeling (i.e., the software process) and to a refinement of the problem to be modeled (initially formulated during the requirements specification activity). As input and sources of information for this activity, process documents are possibly supplemented with interviews with people involved in the process (or with the process "owner"). The created process model describes the artifacts used, the processes or activities performed, and the roles and tools involved. The process model shows which activities transform which artifacts and how information flows through the model.

b: Creation of the influence diagram for describing the relationships between parameters of the process

For documenting the relationships between process parameters, influence diagrams can be used. Influence factors are typically factors that change the result or behavior of other project parameters. The relationships between influencing and influenced parameters are represented in an influence diagram by arrows and + or -, depending on whether variation of the factors occurs in the same way or in opposite ways.

When the influence diagram is created, the inputs and outputs identified in the requirements phase should be considered. These inputs and, especially, the outputs have to be captured in the influence diagrams. Fig. 7.6 presents a small excerpt of a static process model, Fig. 7.7 a corresponding influence diagram.

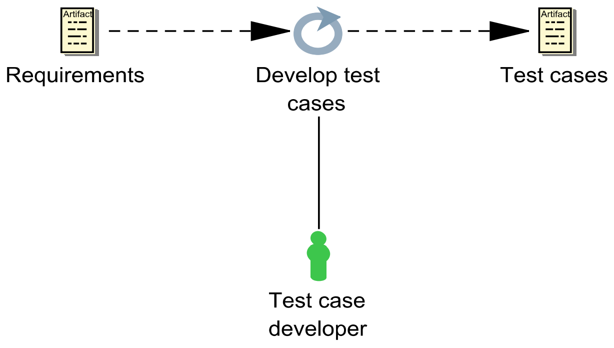
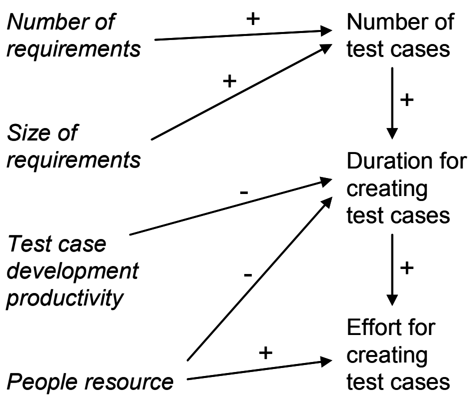


Fig. 7.6: Static process model



Number of requirements
Size of requirements
Test case development productivity
People resource
Number of test cases
Duration for creating test cases
Effort for creating test cases
+
+
+
-
-
+
+

Fig. 7.7: Influence diagram

The influence diagrams that are developed in this step are later refined during design and especially during implementation, driven by a better understanding of what is really needed for implementation.

c: Collection and analysis of empirical data for deriving the quantitative relationships.

For identifying the quantitative relationships between process parameters, one needs to identify which data/metrics need to be collected and analyzed. Usually, not all required data is available, and additional metrics from the target organization should be collected. In this case, developing a process model can help to shape and focus the measurement program of a project.

d: Quantification of the relationships

This is the task that is probably the hardest part of the analysis, because it requires quantifying the relations and distinguishing parameter types. The following parameter types can be distinguished:

- Calibration parameters: for calibrating the model according to the organization, such as productivity values, learning, skills, and number of developers
- Project-specific input: for representing a specific project, such as number of test cases, modules, and size of the tasks
- Variable parameters: These are the parameters that are changed to analyze the results of the output variables. In general, these can be the same as the calibration parameters. The variable parameters during the model's lifecycle are determined either by the scenario from the requirements or by new requirements.

The distinction between these parameters is often not easy and shifts, especially for calibration and variable parameters, depending on the scenarios that are addressed by the model.

The mathematical quantification is done in this step. Depending on the availability of historical metric data, sophisticated data mining methods might be used to determine these relationships. Otherwise, interviews with customers or experts, or literature sources have to be used.

The outputs of the process analysis and specification phase are models (static model, influence diagrams, and relations) of the software development process and parameters that have to be simulated, measures (metrics) that need to be received from the real process, and a description of all the assumptions and decisions that are made during the analysis. The latter is useful for documenting the model for later maintenance and evolution.

The artifacts created during process analysis and specification have two distinct properties: the level of static or dynamic behavior they capture, and the quantitative nature of the information they represent. Fig. 7.8 shows the values for these properties for the static process model (which is static and qualitative), the influence diagram (static and qualitative), and the end product simulator, which is quantitative and dynamic.

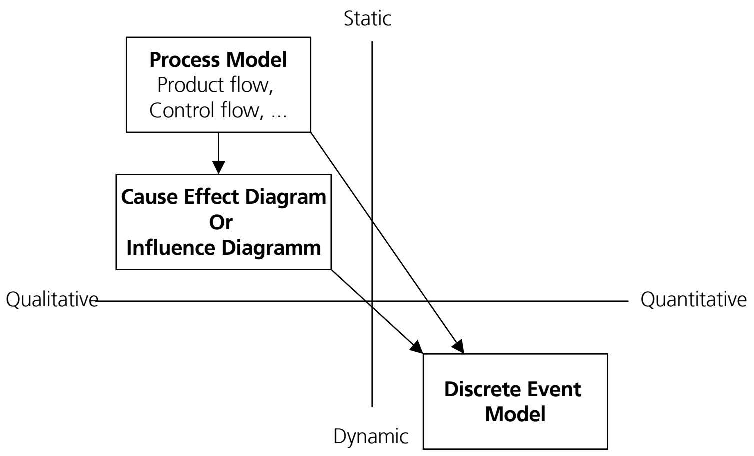


Fig. 7.8: Properties of different modeling artifacts

Throughout this activity, several verification and validation steps must be performed:

- The static model has to be validated against the requirements (to make sure it is within the scope, and also that it is complete for the goal stated in the requirements) and against the real process (here the customer should be involved).
- The parameters in the influence diagram must be checked against the metrics (to ensure they are consistent) and against the input and output variables of the requirement specification.
- The relations must be checked against the influence diagrams for completeness and consistency. All the factors of the influence diagram that influence the result have to be represented, for instance in the equation of the relation.
- The influence diagrams (possibly also the relations) must be validated by the customer with respect to their conformance to real influences.

– The units of measurement for all parameters should be checked.

7.3.3 Model design

During this activity, the modeler develops the design of the model, which is independent of the implementation environment. The design is divided into different levels of detail, the high-level design and the detailed design.

7.3.3.1 High-Level Design

In the high-level design, the surrounding infrastructure is defined. Also, the basic mechanisms describing how the input and output data is managed and represented is defined, if necessary. Usually, a model's design comprises components such as a database or a spreadsheet, a visualization component, and the simulation model itself, together with the information flows between them.

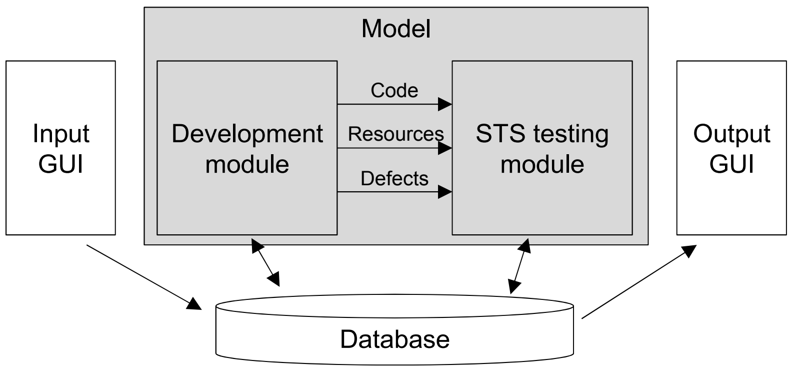


Fig. 7.9: High-level design (excerpt)

Fig. 7.9 shows the high-level design for a system testing simulation (STS) model. The model has two main modules, the *Development module,* which models the development of the software, and the *STS Testing module*, which models the system testing of the software. These two modules interact through the flow of items such as *Code*, *Resources*, and *Defects*. The whole model has graphical user interfaces (GUI), both for input and for output. Through the input interface, the user of the simulator provides the values of the input parameters, which are saved in a database and then fed into the simulator. The outputs of the simulator are also saved in the database and then used by a visualization software, which displays them in a user-friendly format to the users of the model. The high-level design is the latest point in time for deciding which type of simulation approach to use (e.g., system dynamics, discrete event, or something else).

7.3.3.2 Detailed Design

In the detailed design, the "low-level design" of the simulation model is created. The designer has to make several decisions, such as:

- Which activities do we have to model? (i.e., what is the level of granularity for the model?)
- What items should be represented?
- What are the attributes of these items that should be captured?

Additionally, the designer has to define the flow of the different items (if more than one type of item is defined, also the combination of items). When creating the detailed design, the static process model can be used for identifying the activities and items, whereas the influence diagrams can be used for identifying the items' attributes.

The outcome of this activity is a detailed design of the model. Fig. 7.10 shows such a design using a UML-like notation. The *Activity* object models an activity in a software process. Its attributes are: duration, cost, effort, inputs, outputs, resource type, and number (both people and equipment resources). The *Resource* object corresponds to the real-world resources needed to perform an activity. An example of instances of the *People_Resource* sub-class of *Resource* would be *Developer* or *Tester*. The human resources are characterized by their productivity, represented as the attribute *Productivity*. The object *Artifacts* captures the code (*SW_Artifacts*) and test cases (*Testing_Artifact*). The code has as attributes its size, complexity, and number of defects, while the test cases have an attribute related to their capability of detecting defects in code (*Effectiveness*).

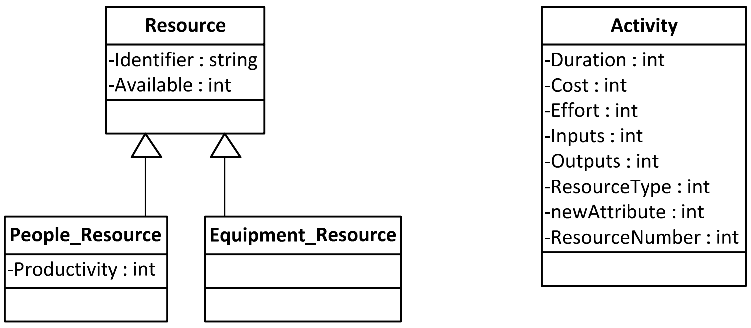


Fig. 7.10: Example of detailed design objects

In the design phase, verification and validation activities must be performed, for example to check the consistency between high- and low-level design as well as with the process description (static, influence diagrams, and relations) and the model requirements.

7.3.4 Model implementation

During implementation, all of the information as well as the design decisions are transferred into the simulation model. The documents from the process specification and analysis and the design are necessary as inputs. This activity in the development process depends heavily on which simulation tool or language is used and is very similar to the implementation activity for a conventional software product. Fig. 7.11 shows an excerpt of a discrete-event model developed in the commercial modeling tool *Extend*, using building blocks connected through inputs and outputs to determine model behavior.

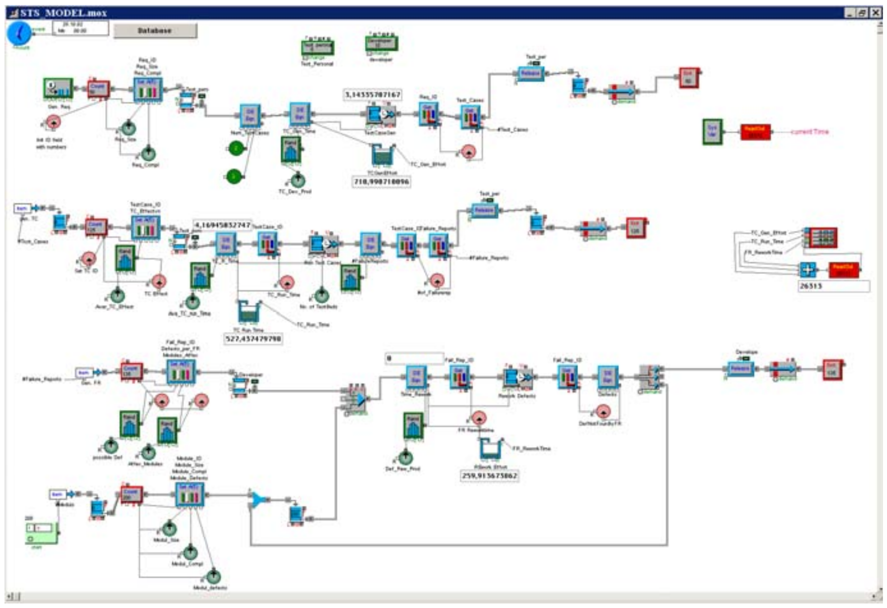


Fig. 7.11: Example simulation model

7.3.5 Model calibration, validation, and verification

Besides the validation and verification that occur throughout development, the implemented model and simulator must be checked to see whether they are suitable for the purpose and the problems they should address. Also, the model will be checked against reality (here the customer/user has to be involved). During such calibration runs, the model's parameters will be set. Fig. 7.12 displays the results of one calibration run. It shows the mean deviation of the model outputs compared to the real experiment it is tested against for different parameter values. It can be observed that for the particular parameter displayed, model deviation from reality is lowest for a parameter value of 0.6 to 0.7.

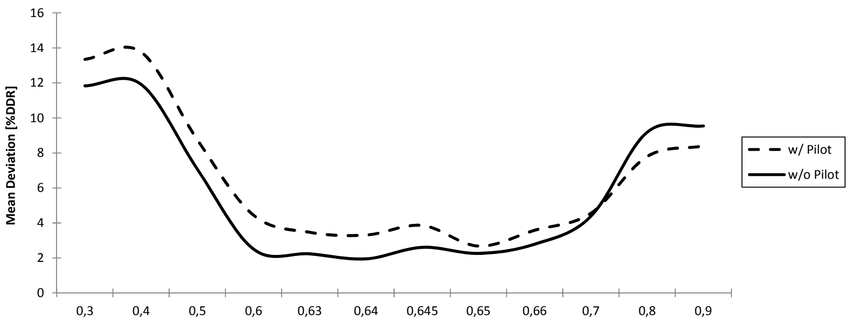


Fig. 7.12: Mean deviation simulation – experiment

Throughout the simulator development process, verification and validation (usually implemented by means of reviews) must be performed after each activity. Also, traceability between different products created during the simulator's development must be maintained, thus enabling the modeler to easily go back to correct or add elements during model development or evolution. Implementation decisions must be documented to allow future understanding of the current implementation and to facilitate model evolution.

Finally, when the simulation model has been validated against reality and calibrated, it can be used to replace real experiments and predict their outcome. Fig. 7.13 displays a plot of one simulation run, depicting the number of defects found during an inspection in relation to the inspection team size. It can be observed that, for instance, with more than five inspectors, the number of defects found during the inspection rises only marginally, whereas the overall effort and the inspection duration continue to climb significantly [12].

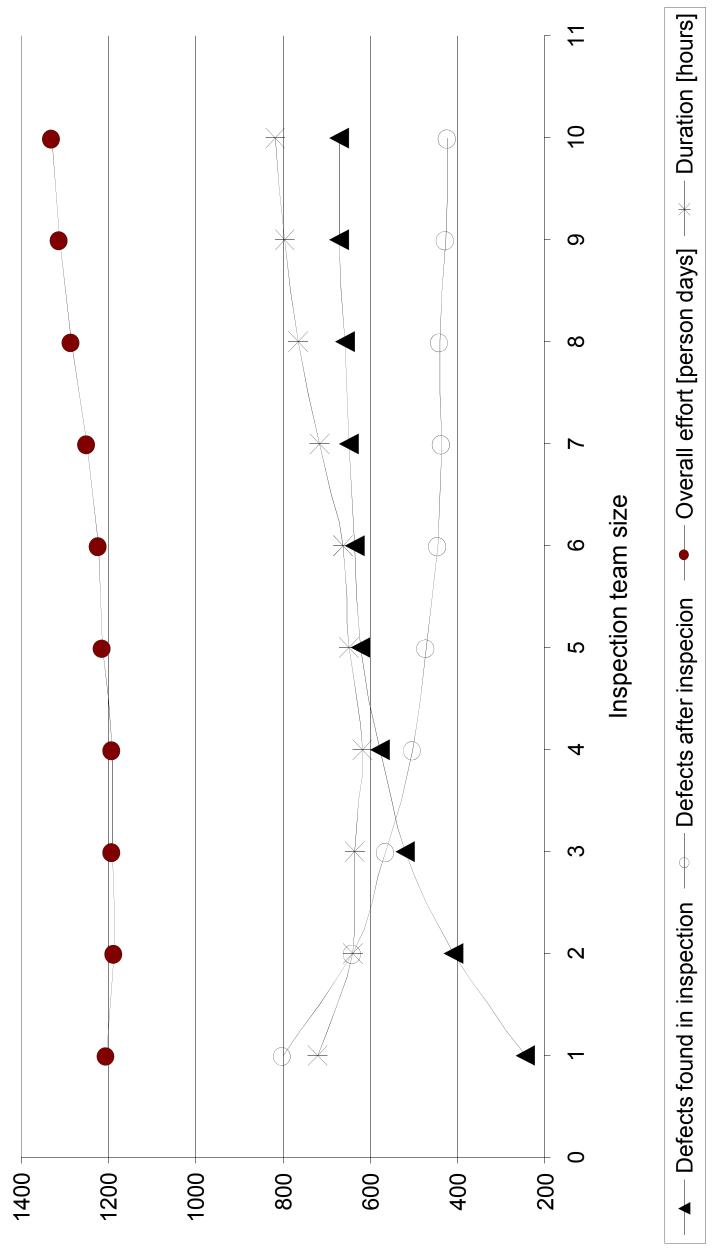


Fig. 7.13: Simulation model run results

7.4 Plug & play process models

With the increasing popularity of software process modeling, it became apparent that process models cannot be developed from scratch every time such a model is needed. However, similar to software itself, a number of patterns can be observed within software processes, for example, a *work-test-rework* pattern that describes the common situation that some work is performed, its results are evaluated by a verification/validation step, and based on the evaluation results, some rework becomes necessary. Such generic patterns can be used for different processes (e.g., requirements elicitation, writing code) and in different domains (e.g., automotive, information systems). It is therefore feasible and recommendable to define such patterns for simulation models as well, forming "macro blocks" that can be used to quickly develop simulation models.

One such approach is the SimSWE library of reusable components for software process simulation [13]. The library provides (tool-independent) definitions of a multitude of components and, for each component, a Matlab®/Simulink® reference implementation under the LGPL license [14] for download [15]. The library contains components for both continuous and discrete-time simulation models. A basic set of components can be used with a standard Matlab/Simulink configuration without add-ons, supporting continuous simulation only. An extended set contains components that can also be used for discrete-event simulation, but requires Simulink addons (at present Stateflow® and SimEvent®). Each set is divided into six sub-collections that represent different types of model components:

- generators, e.g., for requirements,
- estimators, e.g., using COCOMO II [16],
- generic activities, e.g., performing work,
- generic processes, e.g., the work-test-rework pattern mentioned earlier,
- management components, e.g., for employee training, and
- utilities like conversions.

The library is developed and shared as Open Source, yet the license allows for using components in a proprietary context without the need to publish modifications as Open Source as well. The library maintainers encourage users to apply the components and contribute additional components as well, in order to construct a set of standard software process simulation components that allow for the rapid development of software process simulation models and that make them easier to compare.

7.5 Combining process simulation and empirical studies

Even though discussed in separate chapters in this textbook, empirical studies and process simulation models can be combined to get additional benefits. For example, while in the 1960s or 70s, nuclear weapon tests were a rather common phenomenon, this has ceased completely in the new millennium. In part, this reduction has been caused by increased knowledge about the harmful side effects of nuclear weapons (namely radiation and contamination of vast areas for many years), but not entirely.

Since the military has priorities other than preventing ecological damage, but an increasing fraction of human society was no longer willing to accept that, other ways for testing these weapons had to be found. Today, most nations have the ability to simulate nuclear explosions. This saves enormous expenses, because real testing is always destructive and nuclear weapons are not exactly cheap. In addition, simulations are not making people upset. Another great advantage is that simulations can be repeated as often as desired with any set of parameters, which is impossible with real tests.

Other areas where simulation is used amply are mechanical engineering and construction. Here, simulation helps to save cost and time. As a rather young profession, software engineering has only recently started to discover and use the benefits of simulation. Empirical software engineering is not very common yet in industry. Many decisions are still based on intuition rather than on measured data. This trial-and-error approach is very expensive and delivers lower-quality products than more systematic approaches would. When implemented, experiments already yield good results, for example in choosing a reading technique for inspections.

Still, the required experiments cost a lot of money, which companies naturally dislike, despite the benefits. But as in other professions, simulation can also help with this aspect in software engineering. A simulation is conducted in two basic parts: modeling the real-world situation, and afterwards simulating it on a computer. The model tries to reproduce some aspects of the real world as accurately as needed, in a representation that can be used in the simulation. The optimal model would contain exactly those entities and relations needed for the simulation, nothing more and nothing less.

An entity represents an element of the real world, e.g., a person or a document. Like their real counterparts, entities interact with each other; this is mapped to relations in the model. One problem in modeling is that it is usually not clear which entities and relations of the real world need to be modeled. This depends on the scope of the model: If the goal is to predict product quality, other factors must be included than when process optimization is aimed at. In any case, including too many factors increases model complexity unnecessarily and may even influence the results, whereas considering too few factors may render the results unreliable.

Working with simulations can be seen as research in a virtual lab (Fig. 7.14). Conventional software engineering laboratories explore the research object by using various empirical studies, e.g., controlled experiments or case studies. The virtual laboratory examines its research object with simulations. Transferring information from one to the other can help to improve research results on both sides.

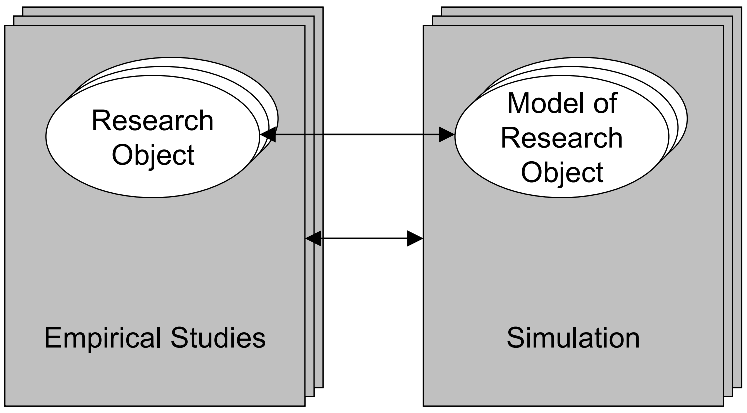


Fig. 7.14: Real and virtual laboratory approach

Simulation supports any of the three types of experiment regarded earlier, each in a different way. Once the simulation model has been built and verified, laboratory experiments can at least partially be replaced, for example to determine the consequences of (gentle) context changes. Case studies focus on monitoring real-life variables, so completely replacing them with simulations would not make sense. Simulations may still be useful for identifying variables that should be monitored. A simulation of an altered process can reveal variables that are likely to change significantly, so the case study can pay special attention to them. In terms of the other direction of the information flow, case studies can deliver valuable data for calibrating the simulation model. The authentic nature of the data (they are taken directly from a real project) also helps to refine the model.

Simulation does not seem very useful for replacing surveys, since surveys are optimized for finding relations in real-life processes. Replacing surveys of real processes with surveys of modeled processes would not save time or money. Actually, surveys can supply information for simulation models, just like case studies: In retrospective, problems often become obvious that were invisible during the project. A survey reveals them in many cases.

Let us say that a certain product was delivered significantly later than planned, and that this phenomenon did not occur for the first time. A survey reveals that the customer changed the initial specifications of the software several times in cooperation with customer service. However, it took customer service several weeks to communicate these changes to the development team, thus already delaying the necessary rework and, in addition, making it more complex by forcing the developers to change a more mature piece of software.

The survey reveals this lack of communication within the software development process, and modeling that part of the process could help to understand and solve the problem. Once the model has been built, the data from the survey can be used to calibrate it. Process changes can then be simulated before realizing them in the actual process. This indicates that surveys and simulation should be used as complements to each other, not as alternatives.

7.6 References

[1] Navarro EO (2006) SimSE: A Software Engineering Simulation Environment for Software Process Education. Dissertation, University of California, Irvine, CA, USA

[2] Kellner MI, Madachay RJ, Raffo DM (1999) Software Process Simulation Modeling: Why? What? How? Journal of Systems and Software 46(2-3):91-105

[3] Abdel-Hamid T, Madnick SE (1991) Software Project Dynamics. Prentice Hall, Englewood Cliffs

[4] Glass RL (1999) The Journal of Systems and Software - Special Issue on Process Simulation Modeling. Elsevier

[5] Fraunhofer Institute for Experimental Software Engineering IESE (1997-2011) Spearmint. http://www.iese.fraunhofer.de/competence/process/pmi/index.jsp. Accessed 27 June 2011

[6] Pfahl D (2001) An Integrated Approach to Simulation-Based Learning in Support of Strategic and Project Management in Software Organisations. Dissertation, University of Kaiserslautern, Germany

[7] Meadows DH, Meadows DL, Randers J, Behrens III WW (1972) The Limits to Growth. Universe Books

[8] Madachy RJ (2008) Software Process Dynamics. Jon Wiley & Sons, Hoboken, NJ, USA

[9] Martin RH, Raffo DA (2000) A Model of the Software Development Process Using both Continuous and Discrete Models. Software Process: Improvement and Practice 5 (2-3):147-157

[10] Rus I, Neu H, Münch J (2003) 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), May 3-4, 2003, Portland, OR, USA

[11] Basili VR, Caldiera G, Rombach HD (1994) Goal Question Metric Paradigm. John Wiley & Sons

[12] Neu H, Hanne T, Münch J, Nickel S, Wirsen A (2003) Creating a Code Inspection Model for Simulation-Based Decision Support. In: Proceedings of the 4th International Workshop on Software Process Simulation and Modeling (ProSim), May 3-4, 2003, Portland, OR, USA

[13] Birkhölzer T, Madachy R, Pfahl D, Port D, Beitinger H, Schuster M, Olkov A (2010) SimSWE - A Library of Reusable Components for Software Process Simulation. In: Proceedings of the International Conference on Software Process (ICSP 2010), Paderborn, Germany, July 8-9, 2010: 321-332

[14] Free Software Foundation (2007) GNU Lesser General Public License. http://www.gnu.org/licenses/lgpl.html. Accessed 27 June 2011

[15] Birkhölzer T, Madachy R, Pfahl D, Port D, Beitinger H, Schuster M, Olkov A (2010) SimSWE - Library for Software Engineering Simulation. http://simswe.ei.htwg-konstanz.de/wiki_simswe/index.php/Main_Page. Accessed 9 June 2011

[16] Boehm BW, Harrowitz E (2000) Software Cost Estimation with Cocomo II. Prentice Hall International