International Journal of Digital Information and Wireless Communications (IJDIWC) 1(2): 346-365
The Society of Digital Information and Wireless Communications, 2011 (ISSN: 2225-658X)

Using the BPM Method and System Dynamics for Simulation of Software Processes

Štěpán Kuchař, Jan Kožusznik, David Ježek, Svatopluk Štolfa
Department of Computer Science,
Faculty of Electrical Engineering and Computer Science
VŠB - Technical University of Ostrava
708 33, Ostrava - Poruba, Czech Republic
{ stepan.kuchar, david.jezek, jan.kozusznik, svatopluk.stolfa }@vsb.cz

ABSTRACT

Modeling and simulation of a software process is one way a company can decide which software process and/or its adjustment is the best solution for its current project. Since there are many different approaches to modeling and simulation and all of them have pros and cons, the very first task is the selection of the appropriate and useful model and simulation approach for the current domain and selected conditions. In this paper we focus on applying a discrete event based modeling and simulation approach and system dynamics modeling and simulation approach to the real case study of the software process. The issue is the comparison of the approaches that should answer the questions: what type of information can we get from the simulation results and how can we use it for decisions about the software process.

KEYWORDS

System dynamics, Petri nets, BP Studio, BPM Method, Software process simulation

1 INTRODUCTION

Business processes represent the core of company behaviour. They define activities which companies (i.e. their employees) perform to satisfy their customers. For a definition of the term *business process*, we use the definition from [1]: "*Business process* is a set of one or more linked procedures or *activities* which collectively realize a business objective or policy goal, normally within the structure defining functional roles and relationships." A process of IS development is called *the software process*. *The software process* is also a kind of business process but it has its specific aspects[2].

Software engineering is a discipline that is involved in software process development and maintenance [3]. The main purpose is risk and cost reduction during software development.

A process development could be divided into activities of different kinds as is defined by the process life-cycle engineering [4]. The process life cycle spiral includes these activities: M*eta-modeling*; *Modeling, Analysis; Simulation; Redesign; Visualization*; *Prototyping, walk-through and performance support; Administration; Integration; Environment generation*; *Instantiation and enactment; Monitoring, recording and auditing; History capture and replay; Articulation; Evolution*; *Process asset management.* Description of every activity is provided in [4].

Since we are cooperating with local software development companies on the development of the best practices for the software development, our intention was to use modeling and simulation tools for the examination of software processes in companies. Although we know, that there are many approaches and tools [5], at the very beginning we just wanted to use our own business modeling tool called BPStudio [6,7]. BPStudio is a discrete event based modeling and simulation tool that was primarily developed for the modeling and simulation of classic business processes. We have created some models in that tool and soon realized, that the software process is actually very specific. The use of our tool was limited to some extent of getting some of the desired information from the simulation. We have realized that the discrete event based simulation cannot really fulfill all our expectations and we decided to involve some other approaches.

In this paper we are going to describe an application of a discrete event based modeling and simulation approach and a system dynamics modeling and simulation approach to the real case study of the software process. The goal is a comparison of the approaches that should answer the questions: what type of information can we get from the simulation results and how can we use it for decisions about the software process.

This paper is organized as follows: section 2 describes the ways of software process modeling and simulation; section 3 presents our tool and method for discrete event based modeling and simulation; section 4 summarizes the system dynamics approach for modeling and simulation of software processes. In section 5 we present the case study of the software process, its modeling and simulation using both approaches and we discuss result. Section 6 concludes and discusses the future work.

2 SOFTWARE PROCESS MODELING AND SIMULATION

There exist many modeling techniques for process modeling as is mentioned in [8]. On the other hand, the software process is quite specific [9] and it has been characterized as "the most complex endeavor humankind has ever attempted" [10]. However, software process could be modeled formally [11].

The main objectives and goals for software process modeling are defined in [11]:

- facilitate human understanding and communication
- support process improvement
- support process management
- automated guidance in performing process
- automated execution support
- simulation

A simulation is a specific objective for the modeling and it has been used in the last two decades for the software process [12]. For a planned project, it enables calculating or predicting: cost (effort), duration, or its schedule. More specific, main reasons why to simulate is defined in [5]:

- strategic management [13,14]
- planning [14]
- control and operational management [14]
- process improvement and technology adoption [14]

- understanding, [14] and
- training and learning [14]

Generally, simulation helps to achieve the optimal balance among quality, budget and duration [15]. Simulation helps forecast and quantify process trajectories in respect to their actual performance [16].

Here are the leading paradigms used for software process simulation [5]:

- discrete event simulation [17] – controlled by discrete events occurrence, useful for modeling activity orchestration
- continuous simulation System Dynamics [17,18] – controlled by continuous time and change of parameters in process is modeled as system of differential equations
- state-based simulation – not widely used
- hybrid model [19,15] – combines approach of discrete event approach and continuous simulation system dynamics
- knowledge-based simulation[16] - textual and primarily used for process understanding and educational purposes
- agent-based simulation [20] - just starting to be applied to system and software development
- qualitative [21]

3 PETRI NETS AND THE BPM METHOD

Petri Nets are one of the formal mechanisms used for the description, verification and simulation of processes. The classical Petri Nets were founded by Carl Adam Petri [22] as a basic tool to describe chemical processes, but since then they have evolved into very strong process modeling technique that supports temporal aspects of the process, stochastic behavior, hierarchisation of the models and even description of process resources and data (High-Level Petri Nets [23]). Properties of Petri Nets have been studied for over 40 years and this research makes Petri Nets a very well defined mechanism for verification and simulation of processes.

Petri Nets describe the process as a set of transitions (activities, tasks) that are connected with places. A place in the Petri Net can contain any number of tokens that represent available resources or occurrences of appropriate event. Whenever all the places preceding any transition are active (they contain at least one token), the transition can be performed. By firing the transition one token from each preceding place is taken away and at the same time one token is generated to every following place. This changes the state of the process that is described by the number of tokens in each place. Simulation in Petri Nets is a sequence of these atomic state changes and thus corresponds to the discrete event type of simulation.

3.1 Basic models of the BPM Method

Modeling and simulation of software processes and business processes in general presents some specific problems which lead to the creation of specialized modeling methods. BPM Method ([7], [24], [25]) is one of these methods that is based on the Petri Nets approach and is used in the case study presented in this paper. BPM Method looks at the three elemental views of the process – architecture of the process, objects and resources utilized in the process and the

behavior of the process. Each of these aspects is described by one of the models included in the BPM Method.

Functional model identifies the process architecture and its customers and products. The primary focus of the functional model is to answer which processes are cooperating with the main process and which subprocesses are used to perform specific tasks in the main process. This model isn't needed for the simulation because it just describes the overall architecture of the process, not the exact sequence of steps performed in the process.

Object model identifies static structure of all objects and resources that are essential for the enactment of the process. This model captures all workers employed in the process and their communication channels. It also contains information about all artifacts (documents, products, material, etc.) that are manipulated or created in the process. Each worker and artifact can be characterized by various optional attributes. This model is useful for the simulation purposes because it contains important information about the abilities and properties of the workers and artifacts that can be utilized in the simulation.

Coordination model specifies the behavior of the process as a sequence of activities, what resources the activities demand and which artifacts are consumed and produced. Alternative flow in the coordination model is enabled by multiple activity scenarios and concurrency of the activities can also be modeled. Complex activity chains can be substituted by sub processes allowing hierarchisation that helps to clarify the process. This model is based on the Petri Nets formalism and is the most important part of the simulation.

3.2 Coordination Diagram

Each of these three models can be visualized by the appropriate type of diagram – Functional diagram, Object diagram and Coordination diagram. The most important in terms of simulating the process is the Coordination diagram and its basic elements are described in table 1**.**

To provide an example of the Coordination diagram a small part of the process used in the case study presented in this paper is modeled in figure 1**.**

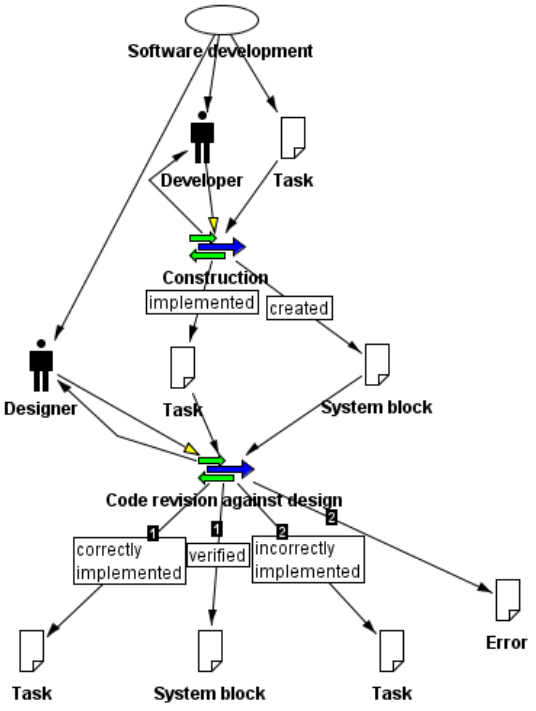


Figure 1. Coordination diagram

This diagram describes a part of the Software Construction subprocess. The first modeling element at the top of diagram shows the interface with the

main process called Software Development. This modeling element allows for the hierarchisation of the model. The Designer and Developer places are the human resources that come from the main process and Task place defines the appropriate artifact. When the Task is created and the Developer resource is available the Construction activity can be performed. By completing this activity the state of the Task changes to implemented, new System block is created and the Developer is ready to implement another task.

The subsequent activity is Code verification that is performed by the Designer and consumes the implemented Task and created System block. This activity can end up in two ways (two scenarios). The first scenario signifies that the constructed code is correct and its outputs are marked by number 1. The second scenario shows that there were errors in the implementation and the process will continue by reporting and repairing the error. Outputs of the second scenario are marked by number 2.

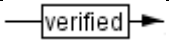
Element	Notation	Description
Active Object	 Developer	Active Object represents any active entity that can perform activities in the process. These objects are mainly human resources but can also be active machines or systems in the process.
Passive Object	 Module	Passive Object is any entity manipulated by the process. These objects are mainly documents, database records, artifacts, source code, etc.
Activity	 Verify Code	Activity is any atomic executable process step that performs some functionality in the process.
Process	 Development	Process elements are used for the hierarchisation of processes because they enable creation of subprocesses and connected processes on the same level of hierarchy.
Flow	 verified	Flow describes required or produced objects of any activity or connected process. It can also specify in which state the object is required or produced.
Activity Scenario	 1	Some activities can be performed in multiple ways often producing different output objects (or objects in different states). Activity Scenarios define which objects are produced by the appropriate scenario of the activity specified by the number on the Flow element.
Responsibility		Responsibility is a special case of the Flow element that specifies which Active Object is responsible for the activity.
Inhibition		Inhibition blocks the start of the inhibited activity while the appropriate object is present.

Table 1. Basic elements of the Coordination diagram (adopted from [6])

3.3 Parallelism in the BPM Method

As you can see in this example, activity scenarios are used to branch the flow through the process but no specific modeling elements for parallelism are contained in the BPM Method. Parallel flows in the BPM Method can be modeled in several ways:

- One activity can be performed concurrently many times, but each such activity instance consumes appropriate input object instances. If there were for example three Developers available and five Tasks created, then the Construction activity in Figure 1 could be performed three times concurrently. Each activity instance would consume one Developer and one Task leaving two Tasks for the Developers that finish their first Tasks.
- Every process instance runs concurrently to other running process instances be they of the same process or of another processes. Each process instance is independent from other instances and only share common resources (for more specific information on shared resources see chapter Software process simulation based on the BPM Method).
- Several different activities in the same process instance can be performed in parallel by structuring the model correctly. The correct structure is based on the Petri Nets *AND-split* where multiple branches start from one transition (activity) and *AND-join* where multiple branches merge in one transition (activity). For example activities Implement Functionality and Create Test in the simplified example process in figure 2 can be performed concurrently because they are both enabled after activity Specify Requirement ends and produces its outputs. At the same time activity Test Functionality can be performed only after both Implement Functionality activity and Create Test activity are completed.

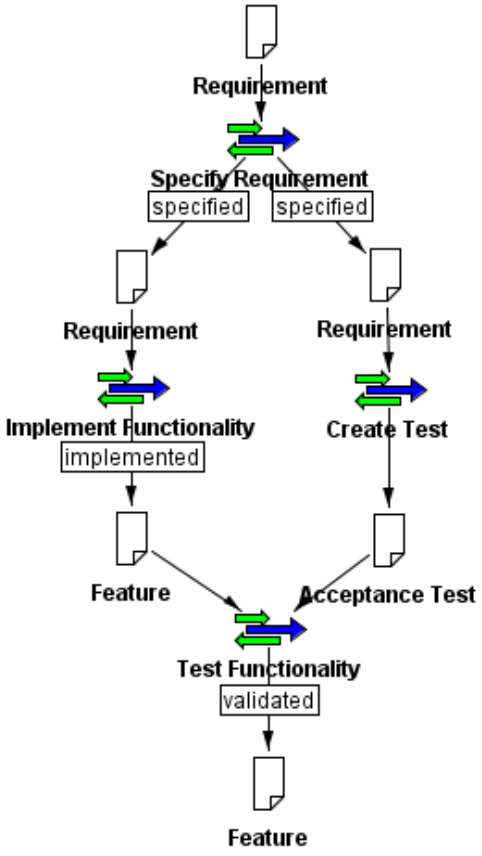


Figure 2. BPM Method parallelism example

4 SYSTEM DYNAMICS

System dynamics model software process as a closed-loop version of a system (figure 3). Input represents requirements, specifications and resources; otherwise output represents artifacts. The input is transformed to output and the operation is influenced by

a controller that changes the behavior of the system.

System dynamic simulation approach was developed by Jay Wright Forrester – published in [26]. It was adopted for the software process modeling purpose by Tarek Abdel-Hamid [12]. This approach is also named continuous system modeling because the model is computed in respect to continuous time. It is the main difference from discrete-event modeling.

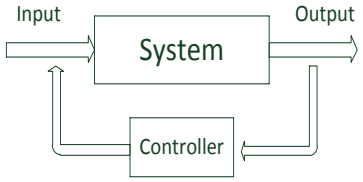


Figure 3. Closed-loop system

Basic elements of the System dynamics model are described in table 2.

The system is modeled as a structure of levels connected by flows and information links. Flows between levels represent process entities that are conserved within flow chains. Information links only provide data from auxiliary, flow or level to another auxiliary or flow (figure 4). Individual events are not tracked; tasks are modeled by levels and the system could be described by system differential equations.

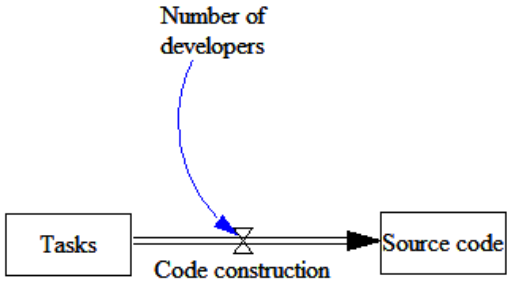


Figure 4. Code construction

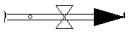
Element	Notation	Description
Level		Level represents an accumulation over time – also called a stock or state variable.
Source/Sink		Source/sink indicates that flows come from or go to the external environment of the process.
Rate		Rate (also called flow) effects changes in levels.
Auxiliary	Only label	Auxiliary is a converter of inputs to outputs, it helps elaborate the detail of level and rate structures.
Information Links		Information Links are used to represent the information flow that could influence value of some auxiliary or rate element.

Table 2. Basic elements of System dynamics (adopted from [27])

It has a very simple parallel to the physical system of water basins (level) connected with valued pipes (rates). Pipes could be controlled based on the level of other water basins or other computed factors (auxiliary).

Information about control is represented by an information link. The amount of level in a specific time T is equal to:

$$\int_0^T \left(\sum_{i \in Input} f_i(t) + \sum_{o \in Output} f_o(t) \right) dt \quad (1)$$

Functions $f_i(t)$ represent input flows to a level, while $f_o(t)$ represents output flows.

The system dynamics approach to software process modeling represents artifacts (requirements, design, source code, documents, errors, tasks) as value of levels – it set this apart from Petri net approach because it does not distinguish any particular artifact. Transformation or artifact production is modeled by flows where time of duration or productivity is expressed by rate of flow.

5 SIMULATION EXAMPLE OF THE SOFTWARE PROCESS

The model of the process consists of eight sub-processes: Requirement analysis, Architectural design, Design, Construction, Testing, Deployment, and Incident management (figure 5).

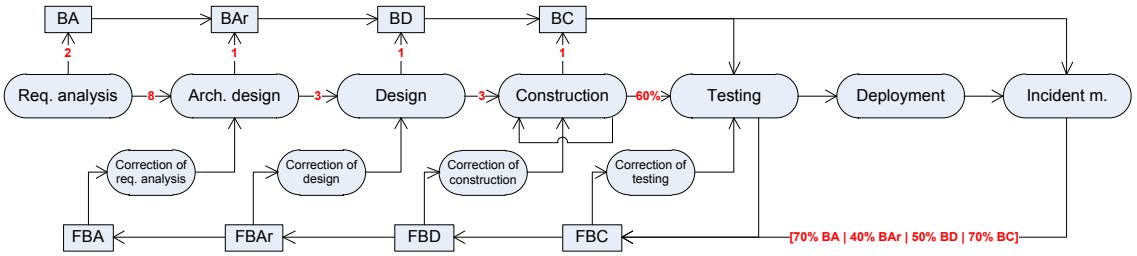


Figure 5. Schema of the process example

The sub-process, "Requirement analysis," contains two main activities: "Make Analysis document" and "Customer review of an Analysis document". The first activity creates eight software features in one step that are the input to the next activity. The second one creates two analysis faults (BA) in one step that simulates the fault rate of this sub-process.

The next sub-process, "Architectural design," consists of one main activity that is executed for each input feature that is generated in "Requirement analysis." Each execution generates one fault in architectural analysis (BAr) and 3 system architecture blocks that are passed to "Design" sub-process.

Each system architecture block from the previous sub-process is processed in the sub-process "Software design". In that sub-process, three programmers' tasks and one design fault (BD) are generated for each input system architecture block.

Sub-process, "Software construction", is one of the most complicated processes in the model. Basically, the software construction processes all programmers' tasks and the result, the source code, is reviewed by the designer.

The activity that represents the revision has two scenarios: one for the successful review which has a probability of 60%

and one for the unsuccessful revision with a probability of 40%. Successful revision generates a source code for the sub-process "Testing" and one fault (BC) that was not found in the revision. The unsuccessful revision returns the task to programmer and the task has to be processed again.

The main purpose of the sub-process "Testing" in the model is decreasing the number of faults. This sub-process is separated into two steps represented by two activities. The first activity is testing and has two scenarios, both of them with a probability of 50 percent: one for failure during the testing and one for the test without any failures. If there is a failure, the activity "Select type of fault" is executed. This activity contains four scenarios, each for one type of fault: fault in requirement analysis (FBA), fault in architectural design (FBAr), fault in design (FBD) and fault in the source code (FBC). Each scenario has a different probability (20%, 10%, 10%, 50%).

The sub-process, "Deployment," contains the activity build module. The input to this activity is the tested source code. The weight of this input is 20-∞, which means the module is built if and only if there exists at least 20 tested source codes or more.

Incident management models and simulates developed software operation during usage of the system. Faults could be identified by the customer – identified fault is named "failure" – and they are reported and fixed. Every kind of fault has a specified probability of identification (BA – 70%, BAr – 40%, BD – 50%, BC – 70%). Every fault is identified as failure (terminology for "bugs" classification is specified in [28]) or it becomes a hidden fault (it is the abstraction that is used for simple simulation of the end of the software project).

6 SOFTWARE PROCESS SIMULATION BASED ON THE BPM METHOD

First we modeled the example of software process presented in the previous chapter using the BPM Method as a representative of the discrete event simulation paradigm. Some requirements, specifics and mechanisms set in the example were not supported by the basic BPM Method so we had to extend it by a number of new mechanisms. These extensions are all supported by various types of Petri Nets and their properties and so they don't disturb the formal nature of the method.

6.1 Stochastic durations and scenario probabilities

The first extension was an addition of stochastic parameters to activities in the Coordination model. Stochastic properties were added to the time duration of activities, their waiting time duration and their scenario probability. These parameters and their conversion to the Petri Nets properties were described in our paper [29]. In this paper we also introduced a very important mechanic for running automatic simulations for modeled processes to test their performance for multiple concurrently running projects. This requirement was fulfilled by adding the stochastic parameter to objects in the Coordination model that defined time intervals between individual entries of new customers to the process.

6.2 Sharing limited resources

But to complete the possibility to run the performance testing simulations another extension was also needed – sharing of limited human and artificial resources in the process. We managed to implement this feature by introducing the pools of limited resources to the objects in the coordination model. Objects with the same shared pool are then linked together and when one of the resources in the pool is used to perform an activity, it is taken from all linked objects. When the activity finishes, the used resource is returned to the pool and thus returns to all linked objects. This behavior can be converted to the Petri Nets formalism using the *fusion places* [30].

Fusion places are special places in Petri Nets that always share all their tokens. When a token is consumed from any fusion place, all linked fusion places lose that token. When a token is added to any fusion place, all linked fusion places gain that token.

Each shared pool in the BPM Method corresponds to a set of linked fusion places that are shared between all concurrent simulation instances of the process. These fusion places are then set as the input and output places for all activities that require shared resources from the appropriate shared pool. figure 6 shows the the Coordination diagram with shared pools and figure 7 depicts the corresponding Petri Net with fusion places. Both Analyst Active Objects in the Coordination diagram are taken from the shared pool of analyst resources. This pool is then represented as a set of two linked fusion places in the Petri Net which serve as an input and output place for the Reuse Architecture and Complete Architecture activities. Standard places for both Analyst Active Objects are still used to ensure correct flow of the process if they were for example activated by previous activities.

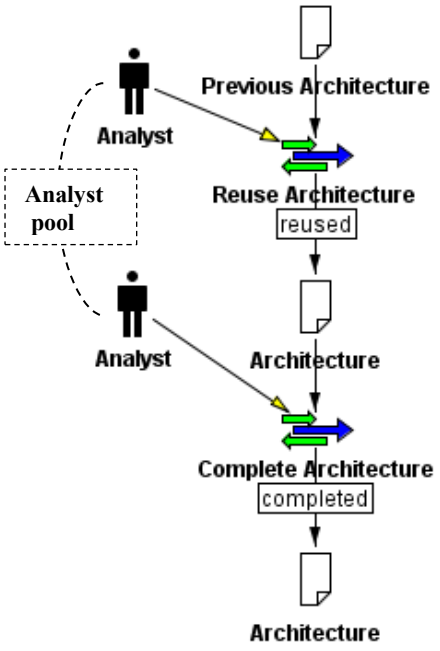


Figure 6. Shared resource pools

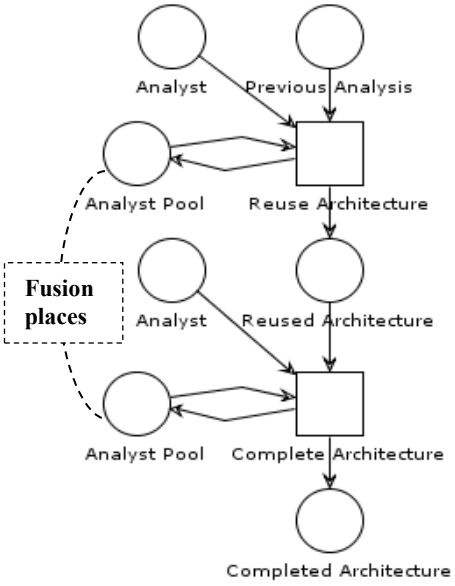


Figure 7. Petri Net with fusion places

6.3 Chained execution

Notation of the Coordination diagram can also describe *chained execution* [31]. Chained execution defines that the same resource is used to perform several subsequent activities in one process instance. In this way the experience acquired by the resource during the first activity can be easily reused in subsequent activities (e.g. when all activities work with the same set of data). The BPM Method models the chained execution by using one object as an output place for one activity and at the same time as an input place for subsequent activity. When this happens the same shared resource will be used for both activities. When the process is modeled as shown in figure 8 and John Smith as an Analyst is chosen to specify the requirements, the same John Smith is then responsible for analyzing these requirements.

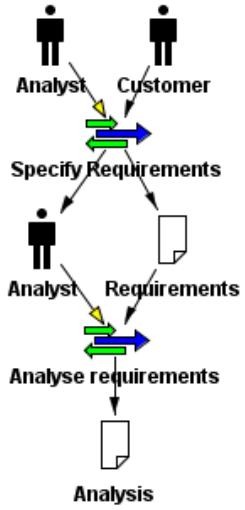


Figure 8. Chained Execution

Chained execution can be easily converted to a Petri Net by adopting some colored properties (Colored Petri Nets [32]), coloring the chained shared resource and then matching the color when choosing the resources for all activities in the chain.

6.4 Weighted flows

The last extension that we added to the coordination model is the notion of weighted flows. Arcs in Petri Nets can be weighted by any natural number and this defines how many tokens are consumed from the input place or produced to the output place when the transition is fired. The BPM Method's flows didn't have this possibility but it is very useful in describing software processes. An example of the weighted flow can be seen in figure 9 where the Release is built only when there are 10 Features ready.

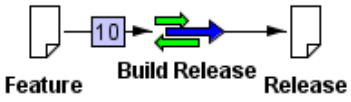


Figure 9. Weighted flows

We implemented all these extensions into the BPStudio software tool [6] used for modeling, simulation and enactment of the processes based on the BPM Method approach and used it for the simulation of the software process example specified in the previous chapter.

6.5 Simulation Experiment

A simulation experiment with the BPM examines one pass through the process. A dependency between a number of work products (tokens in given places) and time is demonstrated in figure 10. These artifacts are inspected: Task (for programmers), Faults (not processed), Failures (discovered fault), Hidden faults (these faults have no chance for

discovering), and Built software (every finished build).

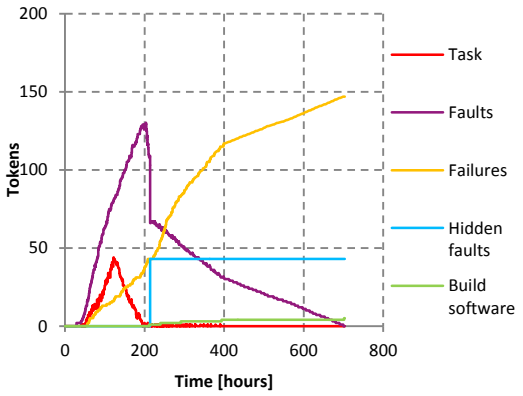


Figure 10. BPM simulation results

A different kind of information is expressed – about a schedule and quality. Three main milestones can be defined in graph – **programmers have started to implement** (50 h) - requirement analysis is finished and a number of tasks starts increasing, when **implementation phase is finished** (210 h) – a number of task stops increasing and keeps near to zero value, because customer operates software and incidents are fixed, and when **software operation is stabilized** (810 h) - number of software builds stops increasing and no other fault could be identified.

BPM simulation use stochastic generator for scenario selection and activity duration, that's make each run of simulation unique. For 500 runs **software operation is stabilized** in range 557h – 1005h with mean in 768h and the total number of failures on end of simulation is in range 110 – 180 with mean in 147.

Software quality is expressed by number of hidden and discovered failures.

6.6 Human Resource Utilization

The BPM Method is also able to describe human resource utilization in the process thanks to the shared resources extensions we added to the method. It is therefore possible to simulate the process with varying number of available human resources and choose the best balance between the number of resources and their utilization. This resource utilization is measured by simple counting up the time when the resource is performing any activity.

Utilization is an interesting result of the simulation but it isn't very useful in optimizing the performance of the process. When optimizing the number of resources in the process we aren't interested in answering how long one resource was doing something in the process, but rather how long did we have to wait for the resource when we needed it to perform another activity. One resource can't perform two activities at the same time but processes run concurrently and they very often need the same resource to be able to continue their run (e.g. one developer is needed to implement a new feature to one system and at the same time needs to repair a fault in another system). When this happens the resource somehow has to perform these tasks sequentially:

- finish the first task and then start the second one, or
- pause the first task and return to it after finishing the second one, or
- switch back and forth between these tasks.

In either way one task will have to wait for the completion of the other (or partial completion in the case of the third

option). It is therefore important to be able to simulate and measure these waiting times. The BPM Method can easily model this, but is able to model only the first sequencing option (i.e. finish the first task and then start the second one). Whenever an activity is enabled but the resource isn't available, the BPM Method counts and notes the time needed for the resource to become available to perform the activity. Total waiting time for one resource is then just a sum of these noted times for this appropriate resource.

To show an example of the utilization and waiting time we made a simulation experiment on the modeled software process. The simulation was set to two customer entries and the second entry came to the process a week after the first. The implementation team consisted of one change manager, three developers, two designers, one project manager, one analyst and one tester. Results of this simulation are shown in figure 11.

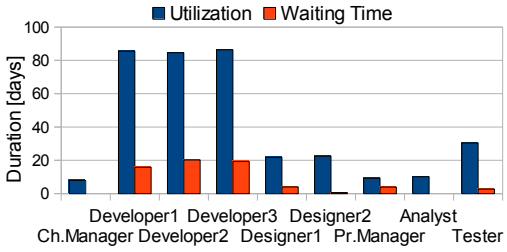


Figure 11. Utilization and Waiting time of human resources

These results show that the most utilized resources are the developers and designers (two designers combined are more utilized than one tester). At the same time, the waiting time shows that this team could profit by hiring another developer because the process had to wait the most for the availability of these resources.

But how much would the process benefit if additional resources were hired? This question can be answered by running simulations with varying number of resources and comparing the utilization results. A following experiment was performed with the same parameters as the former, only the number of developer resources were changed to find the right balance between the number of developers and their utilization. Results of this experiment are shown in figure 12.

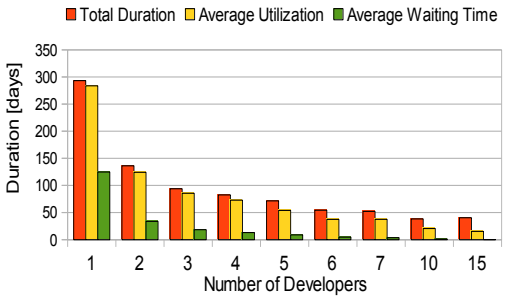


Figure 12. Process performance related to the number of developer resources

The utilizations and waiting times of each resource in one role are very similar as seen in figure 11, so we used the average value to visualize the graph. This similarity is caused by the fact that all resources have the same properties in the BPM Method. But in reality every human resource is different with diverse sets of abilities and skills and this could be integrated into models and simulations.

Results depicted in figure 12 show that the process is highly parallel for a small number of developers, because the utilization of developers is only a bit lower than the total duration of the

process. This property starts to degrade at approximately 5 developers in the team. The next indicator could be the rate of total duration reduction that starts to stagnate at about 6 developers. If we wanted to minimize the waiting time, the ideal number would be 15 developers, but at this point the developers are highly underutilized.

7 SOFTWARE PROCESS SIMULATION BASED ON SYSTEM DYNAMICS

Example implementation with system dynamics is different to the previous due to the mentioned differences. Modeling of specific construction will follow. In advance, system dynamics don't use stochastic parameters. This condition is fulfilled by the substitution of stochastic parameters with their means – activity duration and number of generated artifacts.

7.1 Activities and Artifacts

The number of artifacts is modeled with levels: Required_features, System_architecture_blocks, Tasks, Revised_source_code etc. Activities transforming input artifacts to output ones are represented by two co-flows (p. 165 in [27]) connected to specific level variables. There is an example of an activity named Design in figure 13. This activity represents the creation of the detailed design based on designed system blocks. The detailed design is represented directly by the artifact Task that is processed in the activity Code_construction. The first flow represents processing of input artifacts (System_architecture_blocks created in Architecture_design activity) and is influenced by the mean value of activity duration and the number of participating employees:

$$Design = \frac{employee}{duration} \quad (2)$$

The second flow means the production of a new artifact (Tasks) and its rate depends on rate of the first one and artifact productivity - number of output artifacts created from one input artifact:

$$Design_{production} = Design \cdot productivity \quad (3)$$

In advance, the rate of first flow should be controlled to ensure the nonnegative values of the input level:

$$Design = \begin{cases} 0 \dots if\ System\_architecture\_blocks \leq 0 \\ \frac{employee}{duration} \dots other\ wise \end{cases} \quad (4)$$

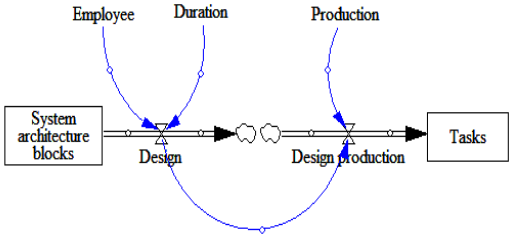


Figure 13. Design

7.2 Fault generation

A very similar concept is applied to the fault generation. There exists one flow for every kind of fault. These flows depend on production of a specific artifact. Productivity means in that case how many faults are produced during the production of output tasks. Faults are aggregated by specific level variables – see figure 14.

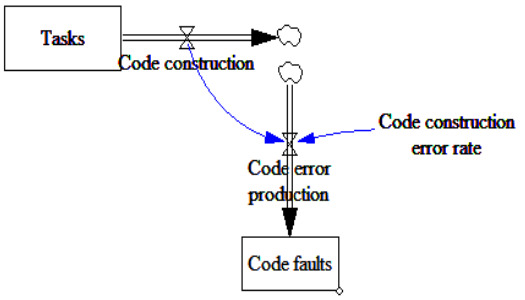


Figure 14. Error production

7.3 Activity Scenarios

Different scenarios are modeled by different flows (situation named split flow process on p. 165 in [27]). The probability of every scenario defines the coefficient that is used for representing flow. There is an example of two scenarios during code revision in figure 15.

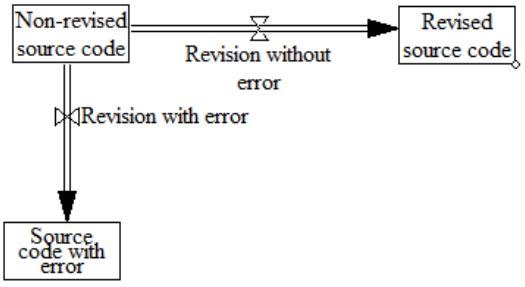


Figure 15. Code revision - split flow process

Each scenario has its own separate flow. Code revision without error has a probability of 60% (p_1) and code revision with error has a probability of 40% (p_2). For simplicity, we consider the same duration for both scenarios. The flow rate is defined for each scenario:

$$code\_revision\_without\_fault = \begin{cases} 0 \dots if\ non\_revised\_code \leq 0 \\ \frac{1}{duration} \cdot p_1 \dots other\ wise \end{cases} \quad (5)$$

$$code\_revision\_with\_fault = \begin{cases} 0 \dots if\ non\_revised\_code \leq 0 \\ \frac{1}{duration} \cdot p_2 \dots other\ wise \end{cases} \quad (6)$$

In the case of a different duration of scenarios (t_1 for scenario without error, t_2 for scenario with error), instead of *duration,* the parameter *average_duration* is used, expressing average duration of one element of production:

$$average\_duration = p_1 \cdot t_1 + p_2 \cdot t_2 \quad (7)$$

The scenario for code revision without faults has a duration of 0.375 hours and the probability of occurrence is 60% while a scenario with a bug has 0.292 hours and a probability of 40%. The Nominal_duration is equal to

$$p_1 \cdot t_1 + p_2 \cdot t_2 = 0.6 \cdot 0.375 + 0.4 \cdot 0.292 = 0.3418 \quad (8)$$

It leads to flow rates:

$$code\_revision\_without\_fault = \begin{cases} 0 \dots if\ non_{revised_{code}} \leq 0 \\ \frac{1}{0.3418} \cdot 0.6 \dots other\ wise \end{cases} \quad (9)$$

$$code\_revision\_with\_fault = \begin{cases} 0 \dots if\ non\_revised\_code \leq 0 \\ \frac{1}{0.3418} \cdot 0.4 \dots other\ wise \end{cases} \quad (10)$$

7.4 Changing productivity of developers

The main benefit of system dynamics use for simulation is the continuous change of simulation parameters. For demonstration purposes, we identify the

productivity of developers during code construction as a changing parameter based on their skills. A course of productivity curve is often named a learning curve and is traditionally defined in terms of unit costs of production. The flow rate for the code construction activity is defined:

$$code_{construction} = \frac{employee}{duration \cdot productivity} \quad (11)$$

The most widely used representation for learning curves is called the log-linear learning curve, expressed as (mentioned on page 230 in [27]):

$$productivity = a \cdot x^n \quad (12)$$

An attribute a is the cost of the first unit, x is cumulative output, and n is the learning curve slope. The slope is related to the learning rate expressed in a percentage (value from interval <0, 1>) and it defines that cost scales by p% for every doubling of accumulative output. After it, for slope is true:

$$n = \log_2 p \quad (13)$$

Cumulative output is equal to the number of produced units of source code increased by one (cumulative output express the number of already produced units).

7.5 Simulation Experiment

We will construct a graph with results for similar artifacts as was mentioned previously. The constructed graph is in figure 16. The results are very similar to the BPM. The main difference is in the progression of the amount of faults and failures, and at the time the implementation ends (the BPM simulation provides 200 while the SD provides 230). The progression of the amount of faults ranges from 0 to 210 hours because the BPM model counts all faults and after the first build of software deployed to the customer (after 210h), some of the faults are categorized as hidden. From this point, the progression of the SD is similar. The SD provides continuous modeling and the amount of build software is greater than zero (float number) much earlier than in the BPM simulation because the BPM uses an integer number and the number of the build software is increased from 0 to 1 after 216 hours. The amount of build software is an important value because a value greater than 0 is a condition for incident management to produce incidents from customers. Reported incidents increase the number failures and the number of tasks.

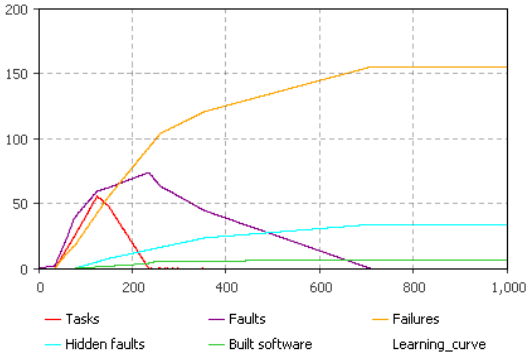


Figure 16. System dynamics result

Different results are displayed when the model of the learning curve is enabled (figure 17). The implementation phase is finished after 170 hours but the software operation is stabilized in almost the same time as in the previous model. It indicates that the productivity of programmers only has an influence on the implementation phase, while it is not

bottleneck according to the duration of the whole project.

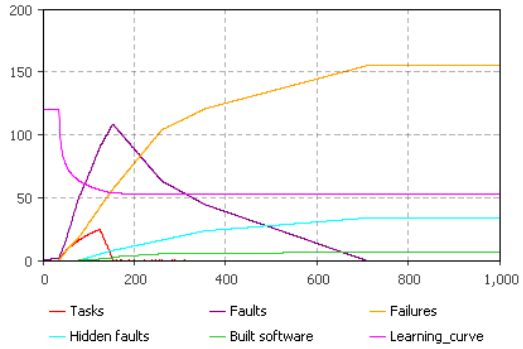


Figure 17. Simulation with learning curve

8 COMPARISON OF THE BPM METHOD AND SYSTEM DYNAMICS

The results of both experiments were similar and both methods proved their abilities in simulating the software process. Regardless, each method has a very different approach to the modeling with its own specific advantages and problems.

The BPM Method has following characteristics:

- Its models exactly describe the real sequence of activities performed in the process and can be easily fine grained by hierarchisation. However this detailed specification can backfire because the processes tend to be very complex and every activity is important. This is particularly relevant for simulations because to acquire accurate results the model has to be very detailed. This can lead to enormous diagrams that are hard to read and manage.
- It is very useful in the management of specific resources in the process and their utilization. Each resource and customer request in the process can have different properties and abilities and this can be used in controlling the flow in the process and choosing the right resource for the job.
- It can very easily sample simulation parameters from any kind of stochastic distributions only when they are needed (i.e. when performing a process step that depends on the parameter).
- The underlying Petri Nets formalism provides the means to verify and enact the modeled process.
- Dynamic parameters (such as knowledge and continuous learning and improvement of human resources, error rate, etc.) and related parameters (such as relation of the workers' skills and activity duration, error rate, etc.) are difficult to describe using the BPM Method.

The System Dynamics method provides following possibilities:

- It can be used on a more abstract level than the BPM Method and doesn't need to be so specific. This is made possible by its continuous approach to everything in the process.
- It is very useful for its feedback options that can influence various related parameters of the process like learning and improvement during the process, communication overhead, error rates and even increasing experience of human resources.
- Everything is continuous, even the creation of all artifacts in the process. This could be used for starting the following parts of the process even when the needed artifact hasn't been finished yet. For example the

implementation phase could start shortly after beginning the design phase, because there is already a part of the design that could be implemented. This is of course possible in the BPM Method as well, but the model has to be specifically designed to behave that way and create partial artifacts to be processed further.

- Discrete process steps and complex sequences of strictly limited activities are difficult to describe using System Dynamics. It is also difficult to implement stochastic behavior to the System Dynamics models.

9 CONCLUSION AND FUTURE WORK

In this paper we described and compared two different approaches to software process simulation – BPM Method as a representative of discrete event simulation and system dynamics for continuous simulations. Even though both approaches can be very effectively used for simulating software processes they vary in the modeling difficulty and they serve different purposes thanks to their inherent advantages. We therefore compared the specifics of these simulation techniques to support the decisions of using one or the other and presented these specifics on the simulation example of the software process. System dynamics turned out better suited for strategic level simulations and BPM Method is more useful in performing simulations on lower, more detailed levels – operational and tactical.

The System Dynamics model of the software process experiment in this paper was designed to behave similarly to the BPM Method model to provide better comparison of modeling possibilities and difficulties. But the model could be enhanced further by additional parameters and by changing existing parameters (delay, error rate, productivity) during the simulation based on actual situation. This was partially shown in the simulation experiment by introducing the learning curve for productivity. By choosing right attributes and their influence on the process the System Dynamics could perform better as a model for supporting and validating strategic decisions and we plan to apply it to the software process and then compare the results with real values of the process.

On the BPM Method side our future research will specialize on one of the greatest advantages of the discrete event models – ability to work with specific instances of resources and their influence on the process. Our next step is individualizing human resources with their skills and competencies and find how their experience changes their productivity and error rate. This could also include skills learning feedback on the human resources level based on the learning-by-doing approach [33].

Acknowledgements. This research has been supported by the internal grant agency of VSB-TU of Ostrava - SP2011/56 Knowledge approach to the modeling, simulation and visualization of software processes.

10 REFERENCES

1. Workflow Management Coalition: Workflow Management Coalition Terminology & Glossary (Document No.

WFMC-TC-1011). Workflow Management Coalition Specification. (1999)
2. Štolfa S., Kožusznik J., Košinár M., Duží M., Číhalová M., Vondrák I.: Building Process Definition with Ontology Background. Paper presented at the CISIM 2010, Krakow, Poland, (2010)
3. Humphrey W.S.: A Discipline for Software Engineering. Addison-Wesley Professional. (1995)
4. Scacchi W., Mi P.: Process Life Cycle Engineering: A Knowledge-Based Approach and Environment. Intelligent Systems in Accounting, Finance, and Management 6:83--107-183--107. (1997)
5. Raffo D.M., Wakeland W.: Moving Up the CMMI Capability and Maturity Levels Using Simulation (trans: Institute SE). (2008)
6. Vondrák I.: Business Process Studio, version 3.0. . VŠB – Technical University of Ostrava, (2000)
7. Vondrák I., Szturc R., Kružel M.: Company Driven by Process Models. Paper presented at the European Concurrent Engineering Conference ECEC '99, Erlangen-Nuremberg, Germany, (1999)
8. Vergidis K., Tiwari A., Majeed B.: Business Process Analysis and Optimization: Beyond Reengineering. Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on 38 (1):69-82. (2008)
9. Raffo D.M.: Modeling software processes quantitatively and assessing the impact of potential process changes on process performance. Ph.D. thesis, Carnegie Mellon University. (1996)
10. Brooks F.P.: No Silver Bullet - Essence and Accidents of Software Engineering (reprinted form information processing 86, 1986). Computer 20 (4):10-19. (1987)
11. Curtis B., Kellner M.I., Over J.: Process modeling. Commun ACM 35 (9):75-90. (1992)
12. Abdel-Hamid T., Madnick S.: Software Project Dynamics: An Integrated Approach. Prentice Hall. (1991)
13. Raffo D., Wernick P.: Software process simulation modelling. Journal of Systems and Software 59 (3):223-225. (2001)
14. Raffo D.M., Kellner M.I.: Empirical analysis in software process simulation modeling. Journal of Systems and Software 53 (1):31-41. (2000)
15. Rus I., Collofello J., Lakey P.: Software process simulation for reliability management. Journal of Systems and Software 46 (2-3):173-182. (1999)
16. Scacchi W.: Experience with software process simulation and modeling. Journal of Systems and Software 46 (2-3):183-192. (1999)
17. Raffo D.M., Vandeville J.V., Martin R.H.: Software process simulation to achieve higher CMM levels. Journal of Systems and Software 46 (2-3):163-172. (1999)
18. Ruiz M., Ramos I., Toro M.: Using dynamic modeling and simulation to improve the COTS software process. In: Bomarius F, Iida H (eds) Product Focused Software Process Improvement, vol 3009. Lecture Notes in Computer Science. pp 568-581(2004)
19. Donzelli P., Iazeolla G.: Hybrid simulation modelling of the software process. Journal of Systems and Software 59 (3):227-235. (2001)
20. David N., Sichman J.S., Coelho H.: Towards an emergence-driven software process for agent-based simulation. In: Sichman JS, Bousquet F, Davidsson P (eds) Multi-Agent-Based Simulation Ii, vol 2581. Lecture Notes in Artificial Intelligence. Springer-Verlag Berlin, Berlin, pp 89-104(2003)
21. Zhang H., Huo M., Kitchenham B., Jeffery R.: Qualitative simulation model for software engineering process. 2006 Australian Software Engineering Conference, Proceedings. (2006)
22. Petri C.A.: Kommunikation mit Automaten. PhD thesis, Institut für instrumentelle Mathematik, Bonn. (1962)
23. Aalst W.M.P.v.d.: Putting high-level Petri nets to work in industry. Computers in Industry 25 (1):45-54. doi:10.1016/0166-3615(94)90031-0 (1994)

24. Vondrák I., Szturc R., Kružel M.: BPM – OO Method for Business Process Modeling. Paper presented at the ISM '99, Rožnov pod Radhoštěm, (1999)
25. Vondrák I., Fedorčák D., Kožusznik J.: Business processes. In Informatics for geoinformatics Conference 2006, Ostrava, 2006.
26. Forrester J.W.: Industrial Dynamics. Pegasus Communications. (1961)
27. Madachy R.J.: Software Process Dynamics. 2nd edn. Wiley-IEEE Press. (2008)
28. IEEE Std 610.12-1990: IEEE Standard Glossary of Software Engineering Terminology. IEEE Standards Board. (1990)
29. Kuchař Š., Kožusznik J.: BPM Method Extension for Automatic Process Simulation. In 8th Industrial Simulation Conference 2010, Lencse G (ed) Budapest, Hungary, June 7 - 9, 2010 2010. Ghent, Belgium.
30. Huber P., Jensen K., Shapiro R.: Hierarchies in coloured petri nets. In: Rozenberg G (ed) Advances in Petri Nets 1990, vol 483. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, pp 313-341. doi:10.1007/3-540-53863-1_30 (1991)
31. Aalst W.M.P.v.d.: The Application of Petri nets to Workflow Management. The Journal of Circuits, Systems and Computers 8 (1):21-66. doi:citeulike-article-id:403742 (1998)
32. Jensen K.: Coloured Petri nets: basic concepts, analysis methods and practical use. Monographs in Theoretical Computer Science, 2nd corrected printing edn. Springer-Verlag. (1997)
33. Hanne T., Neu H.: Simulating human resources in software development processes. In ITWM 64, Berichte des Fraunhofer, 2004. Berichte des Fraunhofer.