SHARING LIMITED RESOURCES IN SOFTWARE PROCESS SIMULATIONS

Štěpán Kuchař
IT4Innovations
VŠB – Technical University of Ostrava
17. listopadu 15/2172
70833 Ostrava-Poruba, Czech Republic
E-mail: stepan.kuchar@vsb.cz

David Ježek, Jan Kožusznik, Svatopluk Štolfa
Department of Computer Science
VŠB – Technical University of Ostrava
17. listopadu 15/2172
70833 Ostrava-Poruba, Czech Republic
E-mail: david.jezek@vsb.cz, jan.kozusznik@vsb.cz, svatopluk.stolfa@vsb.cz

KEYWORDS

BPM method, software process, human-based process, process simulation, Petri nets, shared resources

ABSTRACT

This paper describes the use of the BPM Method simulations for estimating the appropriate number of resources in the software process to balance its cost and duration. Extensions to the BPM Method that are necessary to model the sharing and managing of limited resources are defined and the case study for a software process of a local software development company is presented.

INTRODUCTION

Business processes represent the core of each organization's behavior (Madison 2005). They define a set of activities that have to be performed to satisfy the customers' needs and requirements, roles and relationships of the employees that are needed for actually performing these activities and objects that are consumed or produced by these activities (Šmída 2007). The software process is also a business process that is highly dependent on human creativity, competencies, experience and interaction (Dutoit et al. 2006). Human-based processes tend to be more uncertain and risky than automated processes performed by machines and properly trained employees are one of the main sources of the company's competitive advantage (Hatch and Dyer 2004).

Due to this uncertainty and the scale of the software process it is problematic to exactly predict the duration, cost and quality of the process and to balance these three indicators (Rus et al. 1999). This problem can be solved by running simulations of the process that can work with stochastic parameters and probability distributions to approximately model the uncertainty. After running many simulations their results can be statistically evaluated and can provide an approximation of the duration, cost and quality of the process.

As the quality of the process mainly depends on the capability and maturity of the process (Persse 2006) and competency of its workers (Hatch and Dyer 2004), the duration and cost greatly depend on the number of resources allocated in the process. But how many resources should be allocated to strike a balance between these two indicators? This paper tries to answer this question by running and evaluating simulation experiments for the software process of a local middle-sized software development company.

BPM METHOD

A modeling and simulation method that is able to sufficiently model human-based processes was needed to execute these experiments. For these purposes we used the BPM Method (Vondrák et al. 1999a, Vondrák et al. 1999b) that already provides simulation environment with stochastic parameters (Kuchař and Kožusznik 2010). This method defines three basic models of the process – architecture of the process, objects and resources utilized in the process and the behavior of the process. The most important one of these models for performing simulations is the behavioral model. This model is called the Coordination model and it specifies the behavior of the process as a sequence of activities. It also specifies what resources the activities demand and which artifacts they consume and produce. Alternative flow in the coordination model is enabled by multiple activity scenarios and concurrency of the activities can also be modeled. This model can also be converted to a Petri net to provide exact semantics for performing simulations (Kuchař and Kožusznik 2010).

The Coordination model is visualized by the Coordination diagram and a simple example of this diagram is shown in Figure 1.

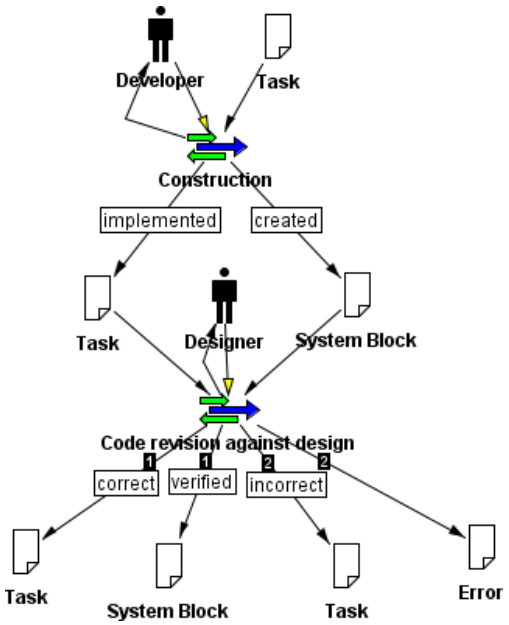

Figure 1: Coordination Diagram

This diagram describes a part of the Software Construction subprocess. The Designer and Developer places describe the roles of employees in the process and the Task place defines the task objects that serve as input for the Construction activity. System block can be constructed when the Task is created and the Developer resource is available. The yellow arrow shows that the Developer is responsible for executing the Construction activity. 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. 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.

PARALLELISM IN THE BPM METHOD

The previous example shows that activity scenarios are used to branch the workflow to a number of alternate branches 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:

1. 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 Tasks first.
2. 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 AND-split structure of the Petri nets where multiple branches start from one activity and AND-join structure where multiple flows merge into one activity.
3. Every process instance runs concurrently with 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 (Kuchař and Kožusznik 2010).

SHARING LIMITED RESOURCES

All of these concurrency methods have to deal with sharing limited number of resources. The first one is directly solved by standard Petri nets behavior because each resource type for one activity is contained in one input place of this activity. When this activity starts all its input resources are consumed and are not available until it finishes. The only extension in the BPM Method needed here was to enable the multiple-server semantics for each activity in the process (Boyer and Diaz 2001).

The second concurrency method can be solved directly by sharing the same input place among all activities in the parallel branches that utilize resources in this place. This will fill the more complex models with a lot of additional arcs but it does not matter for the automatic processing in the simulations.

The third concurrency method is the most problematic because it needs to share resources between multiple instances of the process. This can be implemented 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. To add this behavior to the BPM Method's simulation engine we had to find a way to convert the idea of these pools into the Petri nets formalism to be compatible with the rest of the simulation model. This problem was solved by adopting the notion of fusion places (Huber et al. 1991).

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 both the input and output places for all activities that require shared resources from the appropriate shared pool. Figure 2 shows the the Coordination diagram with shared pools and Figure 3 depicts the corresponding Petri net with fusion places.

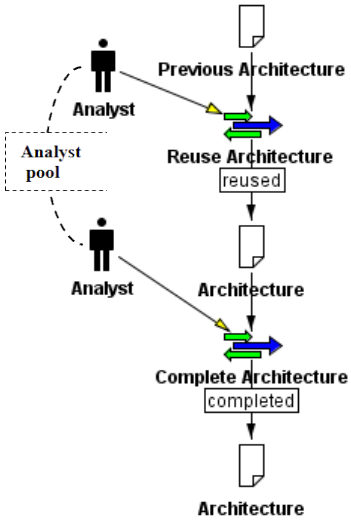

Figure 2: Shared Resource Pools

Both Analyst 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 both input and output place for the Reuse Architecture and Complete Architecture activities. Standard places for both Analyst objects are still used to ensure correct flow of the process if they were activated by previous activities.

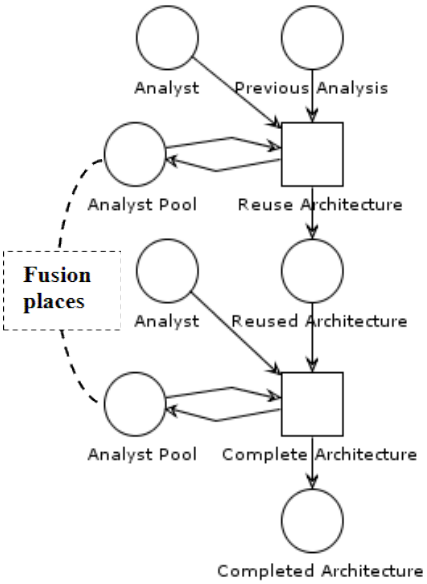

Figure 3: Petri Net with Fusion Places

Fusion places can also be used to solve the second concurrency method instead of the pure Petri net solution mentioned earlier.

CHAINED EXECUTION

Notation of the Coordination diagram can also describe chained execution (Aalst 1998). Chained execution defines that the same resource is used to perform several subsequent activities in one process instance. In this way 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 4 and John Smith as an Analyst is chosen to specify the requirements, the same John Smith is then responsible for analyzing these requirements.

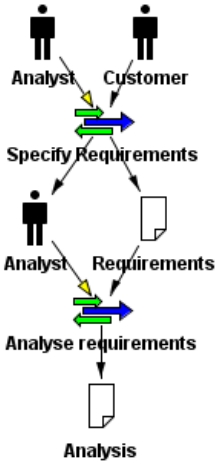

Figure 4: Chained Execution

Chained execution can be easily converted to a Petri net by adopting some colored properties (see Colored Petri nets, e.g. (Jensen 1998)), coloring the chained shared resource and then matching the color when choosing the resources for all activities in the chain.

RESOURCE UTILIZATION AND UNAVAILABILITY

The last extension added to the BPM Method is determining the utilization of each shared resource in the process. This utilization is measured by simply counting up the time when the resource is performing any activity.

Utilization is an interesting result of the simulation but it is not 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 the process have to wait for the resource when it was needed to perform another activity. One resource can not perform two activities at the same time but activities and 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 in one system and at the same time needs to repair a fault in another system). When this happens the resource has to somehow 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. 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 computed by adding up these noted times for this appropriate resource.

CASE STUDY

We have cooperated with a middle-sized software development company from Czech Republic to model and improve their software development processes. Having made some improvements in the process (see (Kožusznik et al. 2011)) a question of how many employees should be allocated to an instance of this process arose. By extending the BPM Method's modeling and simulation software tool called BPStudio with the mechanisms described in previous chapters it was possible to run simulation experiments with varying number of employees and analyze the results to find the best configuration.

The first step was to identify the roles in the process that would influence the total duration of process execution. This can be done by running a simulation with small number of resources in different roles and then running a simulation with more resources. By watching waiting times of these resources bottlenecks in the process can be identified. When the waiting time is high and is changing

with the number of resources then the process could benefit from adding resources to this role. Utilization and waiting times for one of these simulations performed in this case study is shown in Figure 5.

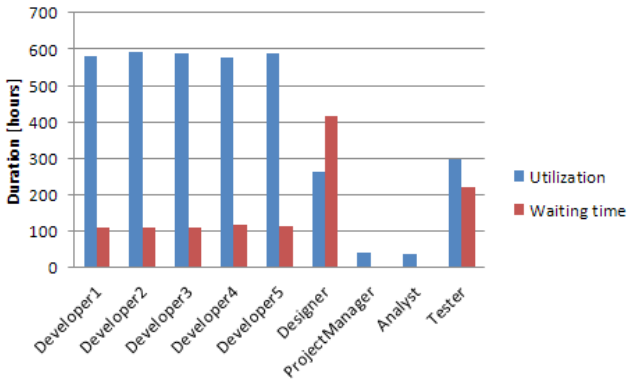

Figure 5: Utilization and Waiting Times of Resources

This simulation showed that only three out of five roles – Developer, Designer and Tester – have significant waiting times so further simulations focused on finding optimal number of employees in these roles. It is also important to note, that all workers of one role have same skills and are therefore interchangeable. This can be also seen on the approximately equal utilization and waiting times of the developers in Figure 5.

The second step was to run simulations with all resource configurations that were feasible in the process. Developers are the most utilized role in the process and by adding more workers their waiting time decreases slowly. This lead to setting the range of possible developers to 2-20. Figure 5 also shows that the lone designer is overburdened with work because designers also play a role of team leaders in this process. Based on this information the range of feasible designers was set to 1-5. The tester has also a lot of unavailable time on his hands but not so much as the designer, so the number of testers was set to 1-4.

After establishing the feasible ranges of workers in each role 200 simulations for every combination of these resource numbers were performed to get the adequate statistical image of the simulations considering their stochastic properties and risks in the process. These combinations were divided to 20 groups with the same number of designers and testers for better understanding and more transparent analysis (i.e. one group was for 1 designer, 1 tester and 2-20 developers; another group was for 1 designer, 2 testers and 2-20 developers, etc.). Finally a statistical analysis was done for each of these groups concerning the duration and total cost of the process. Figure 6 shows the chart with duration results of the group with 2 designers and 2 testers.

The best duration average in this group has the configuration with 20 developers because the waiting time of these developers still hasn't decreased to 0. But statistical comparison of this configuration with the other configurations shows that it is not significantly better than the configurations with 16, 17, 18 and 19 developers (at 95% confidence level). It is therefore better to use the lower number of developers, because such unallocated developers can be sent to work on another project. The chart also shows that starting around the 10 developers mark the slope of the decrease is very small and the process does not benefit much from adding more developers.

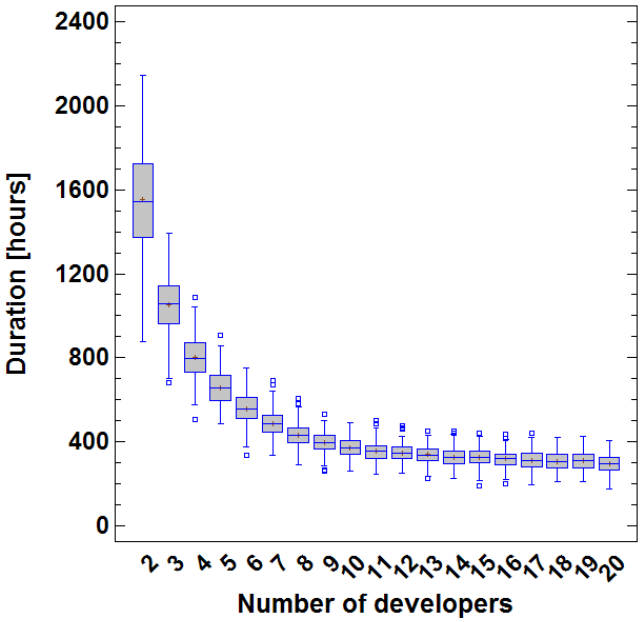

Figure 6: Durations for 2 Designers and 1 Tester Group

This trend is also confirmed by the total process cost chart of the same group in Figure 7. The chart shows that the cost of allocating additional developers to the process is higher then the decrease in the duration for 10 and more developers. The best configuration optimizing the total process cost in this group is 9 developers with 8 and 10 being statistically comparable. This means that 8 developers should be used to free the ninth developer for other projects.

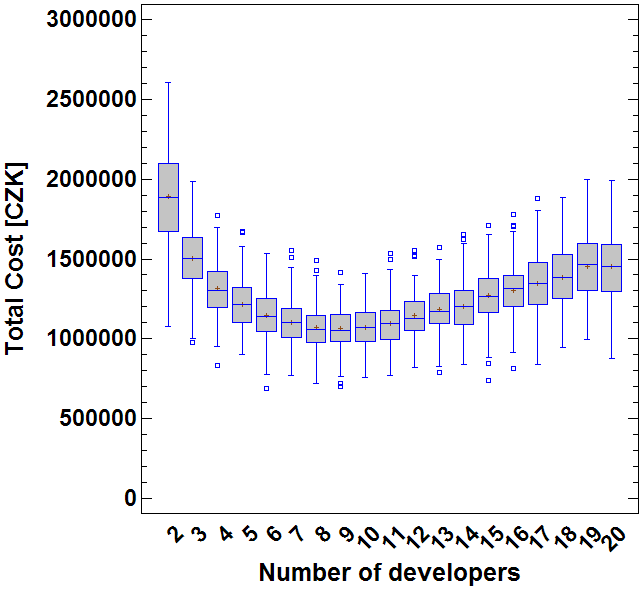

Figure 7: Total Costs for 2 Designers and 1 Tester Group

After analyzing all groups in this manner the best local configurations can be compared with each other in the same way to find the best global configuration that minimizes either the duration or the process cost. The best global configuration minimizing the process duration is 3 designers, 3 testers and 19 developers with 289.08 hours and 1 568 254 CZK. The best global configuration

minimizing the process cost is 1 designer, 1 tester and 8 developers with 637.64 hours and 1 376 698 CZK.

These configurations optimized only one indicator without worrying about the other. But how to find a configuration that balances both? The first step is to normalize average values of both indicators for each configuration in each group by using the following formula:

$$navg(c_i) = \frac{avg(c_i) - min(avg(c_1), \ldots, avg(c_n))}{max(avg(c_1), \ldots, avg(c_n)) - min(avg(c_1), \ldots, avg(c_n))}$$

where n is the number of configurations in actually examined group and c_j for $j \in \{1, \ldots, i, \ldots, n\}$ describes actually examined indicator's values for all simulations of the j-th configuration in actually examined group.

The balancing value of each configuration is then:

$$balance(c_i) = \alpha * navg_{duration}(c_i) + \beta * navg_{cost}(c_i)$$

where $\alpha \in \langle 0,1 \rangle$ is the duration indicator weight, $\beta \in \langle 0,1 \rangle$ is the cost indicator weight and $\alpha + \beta = 1$.

The best balanced local configuration of each group is the one that minimizes this balancing value. These best balanced local configurations can then be collected to one common group and among them the best balanced global configuration can be identified by the same procedure.

The final problem is to correctly set the α and β weights so that they really balance the duration and cost. It is clear that if α=1 and β=0 then only duration is taken into consideration and the configuration with the lowest duration is declared the best. On the other hand if α=0 and β=1 then only process cost is considered in choosing the best configuration. The first idea that comes to mind is setting α=0.5 and β=0.5 so that both are of the same weight but this might not balance these indicators correctly. The balancing value with these weights is very sensitive to duration and cost indicators with different variances. For example if the duration varied from 100 to 900 hours and the process cost only ranged from 1 000 000 to 1 000 010 CZK then the loss of 400 hours would be balanced by the loss of 5 CZK only. That is not a very good ratio even though the Czech economists would like it to be. The solution to this problem is to set the α and β weights in accordance with the range of the duration and cost indicators. This has to be done for each configuration group separately because ranges can be different in each group.

To set the weights correctly a range ratio has to be computed for both indicators that describes how much the maximum average value is greater than the minimum average value by the following formula:

$$ratio = \frac{max(avg(c_1), \ldots, avg(c_n)) - min(avg(c_1), \ldots, avg(c_n))}{min(avg(c_1), \ldots, avg(c_n))}$$

This ratio is then directly used to determine the α and β weights using the following formulas:

$$\alpha = \frac{ratio_{duration}}{ratio_{duration} + ratio_{cost}} \qquad \beta = \frac{ratio_{cost}}{ratio_{duration} + ratio_{cost}}$$

Using these weights the best balanced global configuration came out to be the same as the best duration global configuration – 3 designers, 3 testers, 19 developers.

CONCLUSION AND FUTURE WORK

This paper described a method that uses simulations for estimating the number of resources in different process roles that equally balance the duration and cost of the final process instance. Duration or cost of the process can also be prioritized when computing the balancing value by changing the ratio of balancing weights. This could be used for finding resource configurations that support the need for processes with the shortest possible duration (e.g. for taking advantage of high market demand) or lowest possible cost (e.g. for small start-up projects).

Real processes can also be limited by specific budget or specific time. Such limitations could be easily implemented into the indicated analysis method by filtering out the configurations that exceed these limits.

All simulations in this case study assumed that all workers of one role have same skills and therefore have similar levels of utilization, waiting times and performance. Real workers have different skills that change the way they are allocated. Our future work will be focused on introducing these differences to process simulations.

ACKNOWLEDGMENTS

This research has been supported by the internal grant agency of VSB-TU of Ostrava - XYZ and elaborated in the framework of the IT4Innovations Centre of Excellence project, reg. no. CZ.1.05/1.1.00/02.0070 supported by Operational Programme 'Research and Development for Innovations' funded by Structural Funds of the European Union and state budget of the Czech Republic.

REFERENCES

Aalst W.M.P. van der 1998. “The Application of Petri nets to Workflow Management”. *The Journal of Circuits, Systems and Computers 8* (1):21-66.

Boyer, M., Diaz M. 2001. „Multiple enabledness of transitions in Petri nets with time“. *9th International Workshop on Petri Nets and Performance Models*, 219–228. IEEE.

Dutoit A.H., McCall R., Mistrik I. 2006. *Rationale Management in Software Engineering*. Springer.

Hatch, N. W., Dyer J. H. 2004. “Human capital and learning as a source of sustainable competitive advantage”. *Strategic Management Journal* 25 (12): 1155-1178.

Huber P., Jensen K., Shapiro R. 1991. “Hierarchies in coloured petri nets”. *Advances in Petri Nets*, vol 483:313-341. Lecture Notes in Computer Science. Springer Berlin / Heidelberg.

Jensen K. 1998. “Coloured Petri nets: basic concepts, analysis methods and practical use”. *Monographs in Theoretical Computer Science*, 2nd corrected printing edn. Springer-Verlag.

Kuchař Š., Kožusznik J. 2010. “BPM Method Extension for Automatic Process Simulation”. *8th Industrial Simulation Conference 2010*, Budapest, Hungary.

Kožusznik J., Štolfa S., Ježek D., Kuchař Š. 2011. “Petri Net Based Simulation for SPI”. *17th European Concurrent Engineering Conference/7th Future Business Technology Conference 2011*, London, United Kingdom.

Madison D. 2005. *Process Mapping, Process Improvement and Process Management*. Paton Press.

Persse, James R. 2006. *Process Improvement Essentials: CMMI, Six SIGMA, and ISO 9001*. O’Reilly Media.

Rus, I., J. Collofello, and P. Lakey. 1999. “Software process simulation for reliability management”. *Journal of Systems and Software 46* (2-3):173-182.

Šmída F. 2007. *Zavádění a rozvoj procesního řízení ve firmě*. Grada Publishing, a.s.

Vondrák I., Szturc R., Kružel M. 1999. “Company Driven by Process Models”. *European Concurrent Engineering Conference Proceedings ECEC ’99* 188-193, Erlangen-Nuremberg, Germany.

Vondrák I., Szturc R., Kružel M. 1999. “BPM – OO Method for Business Process Modeling”. *ISM ’99 Proceedings* 155-163, CSSS, Rožnov pod Radhoštěm.