Proceedings of the 2012 Winter Simulation Conference
C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A.M. Uhrmacher, eds

AGENT-BASED SIMULATION OF THE SOFTWARE DEVELOPMENT PROCESS: A CASE STUDY AT AVL

Bojan Spasic

AVL-AST Ltd.
Av. Dubrovnik 10
Zagreb, CROATIA

Bhakti S. S. Onggo

Department of Management Science
Lancaster University Management School
Lancaster, UNITED KINGDOM

ABSTRACT

Software development projects are difficult to manage due to the high uncertainty in their various phases. Simulation is one of the tools that has been used to help software project managers produce project plans. Research into software process simulation modeling (SPSM) shows the dominance of discrete-event simulation and system dynamics. This paper supports the use of ABS in SPSM. We propose a practical effort function to estimate developers' behavior. The other contribution of this paper is to demonstrate how the ABS model can be developed, calibrated and validated using data readily available to many software development companies/departments. This paper focuses on the construction phase of a tailored Rational Unified Process used in a geographically distributed software development department at AVL. The results look promising but more work needs to be done to include ABS into one of the mainstream simulation paradigms in SPSM.

1 INTRODUCTION

DISCLAIMER: *The views and opinions expressed in this article are those of the authors and do not necessarily reflect the official policy or position of AVL.*

Software development projects are inherently difficult to manage. This difficulty comes partly from the complexity of the software development process, and partly from the uncertainty which accompanies all project phases: from inception and planning, when requirements are analyzed and the required effort is under- or overestimated, through to execution, when unforeseen problems and additional effort are revealed, to the non-deterministic nature of software projects when it comes to completion and handover quality. Many organizations manage software development projects using established processes and methodologies to structure their activities. However, an understanding of the intrinsic properties of these processes, as well as the significance and reliability of the metrics used to plan and evaluate particular phases of the process, is often challenged by process complexity. This hinders software project managers' successful planning and affects subsequent monitoring and controlling activities and introduces uncertainty to project scheduling and cost.

Simulation has been recognized as one of the tools that can be used to help software project managers. It has given rise to a field called *Simulation Process Simulation Modeling* (SPSM), which has been around for more than two decades (Zhang, Kitchenham and Pfahl 2008). Simulation of the software development process can provide additional insights which can be useful to gain understanding of the simulated process at the cognitive, tactical and strategic levels (ibid.). In a similar vein, Kellner, Madachy and Raffo (1999) stress the importance of simulation as an aid to decision-making and to set the stage for the characterization of software process simulations through a three-dimensional space defined by the simulation purpose ('why'), scope and result variables ('what') and the methodology employed ('how'). Based on the review by Zhang, Kitchenham and Pfahl (2008), the literature shows that, for the first dimension,

SPSM has been used for prediction and planning, controlling and operational management, risk management, process improvement, technology adoption and process optimization. For the second dimension, the same review shows that SPSM has been used to cover various phases in the software development process which can range from a single phase (e.g. design, coding, testing) to successive releases of multiple products; a simulation can provide such results as effort estimates, schedules, duration, quality levels, resource utilization, productivity and other variables of interest. Finally, for the third dimension, the review shows that the methodology used in SPSM has been dominated by *discrete-event simulation* (DES) and *system dynamics* (SD).

This paper advocates the use of *agent-based simulation* (ABS) in SPSM. Our work is different from much work in agent-based simulation in SPSM because we use real data from a large distributed software development department (henceforth: the Department) at AVL, the world's largest independent and privately owned company for the development of powertrain systems with internal combustion engines, as well as instrumentation and test systems. Software development by the Department and its several hundred developers takes place in development centers located in Europe, North America and Asia. By using real data readily available in practice, we hope that this work can appeal to both practitioners and academics. The model was developed, calibrated and validated using available data. In this paper, we share our experience of putting the theories reported in the literature into practice using available data. The other contribution of this paper is to propose the use of an *effort function* to represent the behavior of a developer when completing his/her assigned task in a project. The remainder of this paper is organized as follows. Section 2 provides an overview of the use of ABS in SPSM. Next, we explain the software development process in the Department. This is followed by a discussion of the modeling work. Finally, we present our concluding remarks in the last section.

2 RELATED WORK

Abdel-Hamid and Madnick (1991) were among the first to publish a book on the topic of simulation modeling in the software development process. This book makes many references to the early work in this area. Based on the number of publications, the research interest in this area has since grown significantly. Recently, Zhang, Kitchenham and Pfahl (2008) conducted a survey to find out what had been done and discussed trends in the field. One of the findings highlights the dominance of DES and SD in SPSM. This is not surprising, given the long history of DES and SD. However, the survey also revealed that there had been an increase in the use of newer simulation methodologies such as ABS in the field.

Wickenberg and Davidsson (2003) were among the first to recognize that the nature of the software development process could potentially be represented in greater detail by an agent-based model (ABM), which could more accurately reflect the interactions between the people, teams or organizations engaged in software development. Modeling individual analysts, developers, testers and other process actors and stakeholders as agents with individual behaviors has the potential to reveal more detail of the process in comparison with the traditional equation-based system dynamics approach. In their work on the development of an ABM to understand open source software evolution, Smith, Capiluppi and Fernandez-Ramil (2006) recognized the importance of qualitative factors such as boredom in their model. Their model was based on the interaction between developers and module agents, where agents' behavior is governed by the balance between developers' boredom levels and the code module fitness index. Dong and Hu (2008) also emphasize the importance of qualitative factors such as technical and social skills competence and the learning ability of developers in their multi-agent based model which aims to understanding the dynamics of software development team effectiveness. The model comprises task, manager and worker agents. The interaction between task process and member agents is described by using the degree of member-task matching, where member competence is compared to task demands. To model the interactions between the members themselves, the authors employ a social relations model. The model is driven by a set of rules governing member-to-member interaction, manager-to-worker interaction, manager selection, worker selection, worker learning and manager-worker rewards.

Given the above account, a key issue often raised with regard to the use of ABS is the need for fine granularity data. One possible data source for ABS in SPSM is historical *personal software process* (PSP) data. This is demonstrated in the work done by Agarwal, who developed a multi-agent based simulation model to simulate an extreme programming (XP)-based software development process (Agarwal 2007; Agarwal and Umphress 2010). Individual developer agents were instantiated using varied Monte Carlo PSP historical data to govern their behavior, and were assigned to user stories defined by user story size, measured in length of code and estimated user story completion time. Another data source includes the various software development data repositories often managed by a number of organizations. For example, Smith, Capiluppi and Fernandez-Ramil (2006) calibrated and validated their model using data obtained from software development data repositories. Data for qualitative factors pose another difficult problem since they are usually not readily available in many software development data repositories. Hence, further data collection using common data collection techniques, such as interviews and questionnaires, is needed. For example, Cherif (2008) collected data from psychological questionnaires to calibrate elaborate agents' behavior in the model.

The next common modeling issue is the choice of metrics to be used as simulation output. This is mainly driven by the modeling objective. Agarwal used the simulated development time for each user story, defect rates, productivity, project velocity and project status at the end of each iteration cycle as the output of his ABS model (Agarwal 2007; Agarwal and Umphress 2010). The objective was to determine whether historical PSP data could be used to predict project metrics. Yilmaz and Phillips (2007) developed an ABS model to assess the efficiency and effectiveness of organizations which employ iterative software development process frameworks such as *Rational Unified Process* (RUP) under the impact of software requirements instability and employee turnover. They used performance metrics such as productivity, staff utilization, timeliness of task completion and the predicted quality of deliverables. Smith, Capiluppi and Fernandez-Ramil (2006) used source code growth metrics in their ABS to show the progress of open source software development. Joslin and Poole (2005) developed an ABS model to produce optimal execution strategies for software development projects which deliver a mixture of mandatory and optional, yet desirable, features. The simulation output included the degree of accomplishment of mandatory tasks, the utility achieved for any optional task that was completed, and the number of resource preemptions (task reassignments).

Finally, one of the most difficult aspects of ABS is validation. The methods reported in the literature vary from no validation (e.g. Joslin and Poole 2005) to the use of a very elaborate method. Cherif (2008) used a very elaborate approach but did not use actual data from previous projects in the validation. Agarwal (2007) validated his model against actual project data and showed high correlation between actual and simulated project duration. Smith, Capiluppi and Fernandez-Ramil (2006) reported having done the validation by comparing the simulation results with the metrics for software evolution of four large open source software systems (system size, proportion of highly complex modules, level of complexity, control work and distribution of changes), obtained by mining software repository data. In their experiments, they varied the possible inputs across the possible values and performed sensitivity analysis. Dong and Hu (2008) validated their model comprehensively by using a virtual situation with an expected outcome, as well as describing an elaborate validation experiment on an actual software development team of 15 members. Yilmaz and Phillips (2007) validated their model qualitatively by comparing it to established observations in several empirical studies of team behavior.

3 SOFTWARE DEVELOPMENT PROCESS

The objective of this research is to demonstrate the use of ABS in SPSM for a real software development process. For the case study, we chose the process used in the Department, which is responsible for the development or enhancement of the software component of large engine/powertrain test bed automation systems. The Department usually splits the development of a software product suite into several software releases. A project is created for each software release. Unless the product suite is new, the software de-

velopment process in each release involves a lot of enhancement and refactoring of existing components, with a comparatively smaller number of new components being added or older components removed.

The current version of the software development process used in the Department is formalized and documented in a set of 27 rules that follows the *Capability Maturity Model Integration* (CMMI) framework. At the time of writing, the software development process is appraised at level 3 ("managed") and consists of six phases: inception, analysis and specification, design and development, integration, test, and stabilization.

The software development process in the Department essentially starts when a stakeholder, typically a product manager, defines a body of high-level ideas or user stories that form the basis for a new suite of product innovation projects. These are in the form of *Enhancement Requests* (ER) and represent the main input for the *inception phase* of each project. ERs may contain a high-level description of an entirely new application, system or platform, or a new feature of an existing application, system or platform that has been recognized by the product manager to have commercial potential. During the inception phase of a new project, its ERs are analyzed, and *feature concepts* (FC) are created. The purpose of feature concepts is manifold: (1) to demonstrate a first level of understanding in terms of software requirements, (2) to provide an overview of possible technical solutions to satisfy the requirements, (3) to provide a first work breakdown structure and effort estimates, and (4) to develop and establish a common understanding of the software to be developed among the process stakeholders.

After a successful review of the deliverables in the inception phase, the process moves into the *analysis and specification phase*. For the execution framework of this phase and the next phase (design and development), the Department has adopted a modified *Rational Unified Process* (RUP) (IBM, 2007) methodology. In this phase, the developed ERs and FCs are handed over to the requirement managers and developer analysts, who will produce two main formal specification documents: a *user requirements specification* (URS) and a *software requirements specification* (SRS). The URS is a hierarchical list of requirements developed using a standard language, in which every requirement presents a distinct functional or non-functional trait of the software to be developed. Each requirement is expressed in a way that makes it possible to construct a question to test if the software fulfills the requirement. The SRS defines how the software fulfills the URS, i.e. how both the functional and non-functional requirements are going to be realized. The realization of the functional requirements is described by a use-case survey. Analyst developers define actors and use-cases according to their expert judgment, their knowledge of the existing software and guidelines, and their understanding of the URS and the ERs. The realization of non-functional requirements (such as performance requirements, testability requirements, security and safety requirements, documentation requirements) is specified by stating concrete conditions that must be fulfilled and verifiable when the software is installed. After the SRS is developed, or in certain other cases, in parallel to its development, a *test case design* (TCD) document is prepared. The document essentially consists of instructions on how to set up and interact with the system under test to validate it against the functional and non-functional requirements specified in the URS and SRS. The instructions are written in the form of test cases, which are designed as questions that evaluate the fulfillment of requirements. A TCD also defines the mandatory set of test cases to be fulfilled in order for a certain type of test to be passed.

During the *design and development phase*, developers create and document the software design (using CASE tools), write code for the components and integrate these into the final application, system or platform. Developers are assigned to work on components based on their skill set and their past involvement with existing or similar components. This is helped by the fact that employee retention in the Department is relatively high. During the course of a project, a single developer can work on one or more components; however, it is possible that one component may require the attention of more than one developer. For example, the visual representation of a UI element may be created by one developer and the logic behind it developed by another, while the developer in charge of integration might need to modify the component for the purpose of fitting it into the overall software package (e.g. creating an INI file, modifying build properties, etc.). Another scenario for multiple developers working on a single compo-

nent is during the code review, which is usually conducted when the first version of a component is implemented. A code review is performed by the peer developer(s) who, sometimes, change the code themselves (instead of sending the code back to the component author), in order to reduce the communication effort between reviewer and author. The design and development phase is concluded when all components have been developed and unit tested, the code has been peer-reviewed, the software design has been approved by the system architect, and all the deliverables necessary to build and integrate the software have been stored in the software repository. Before the software is handed over for system testing, a trial build and integration is performed to create a full installation package, depending on the type of software developed, as this can be a part or whole application, system or platform. A test of the whole software package is performed according to the test case design developed in the specification phase, as well as a "smoke test" of overall software functionality. This test represents the quality gate for the design and development phase: until it is passed (all mandatory test cases are fulfilled), the software development process remains in this phase.

The *integration phase* is needed when a system or platform is implemented in multiple concurrent projects. The deliverables of those projects must be integrated to make one product suite delivery. To facilitate this, an integration test is done. For a project that adds new components to or modifies existing components in an existing system, the test is done by integrating the deliverables with the old system (usually the version before development started, which is also used as the code base) to serve as the platform to test their new functionality before they are merged with deliverables from other projects in the same product suite. This, so-called "pre-merge test" must be passed before the project deliverables are allowed to enter the main release.

Once all contributing software has entered the main release of the system or platform, the process enters the *test phase*, in which the integrators hand over the main release to the testers who will perform the system test. This involves the execution of test cases developed for the new software, as well as a battery of test cases considered to be 'standard'. The standard test cases include overall stability tests, stress tests, performance tests, basic functionality tests and regression tests. During the test phase, the developers communicate with the testers through a defect tracking mechanism. When a software defect is detected, a tester will notify the developers of the defect. Any defect usually pre-empts the developers' other activities (in this phase the developers might already be partially or fully engaged in other projects).

Finally, upon successful completion of the system test, the software is ready to be handed over to the corresponding product manager. In this *handover phase*, the project manager is responsible for organizing, gathering and presenting the project artifacts to the stakeholders and for dealing with the necessary administrative activities to close the project.

4 SIMULATION MODELING

This section explains the data collection process, the simulation modeling process and validation of the simulation model.

4.1 Data Collection

Ideally, we would like to use PSP data to calibrate our model. Such data are not commonly available in a production environment, including in the Department. Organizations are sometimes reluctant to collect PSP data for a number of reasons including cost and potential privacy issues (Johnson et al. 2003). In the absence of PSP data, we collected data from the historical *project database* containing performance metrics from past projects, the *software repository* which stored data from the version control system, and the *booking database* which contained historical data for the actual booking by project team members.

We managed to retrieve data for 300 past projects from the project database. Each dataset contains information about project duration, estimated and actual effort spent on a project, the product suite to which the project contributed, and the architecture and technology employed. The software repository had a massive 5.5 million records containing information such as versions of all source code files and their re-

quirement, specification and design documents. The booking data store detailed information about all checked-in files from relevant projects' repositories, such as timestamp, username, path, file name and type, data size and version number. As reported in many simulation projects, the available data were not perfect, especially because these data sources are independently maintained by different departments for different purposes. Given the quantity of data and the way the data were collected, it was virtually impossible to cleanse the data manually. Hence, we designed a relational database, as shown in Figure 1, so that we could establish and enforce the relationship between the data from originally unrelated sources. This relational database was needed to help us with the data cleansing process and to retrieve the data needed for the simulation modeling.

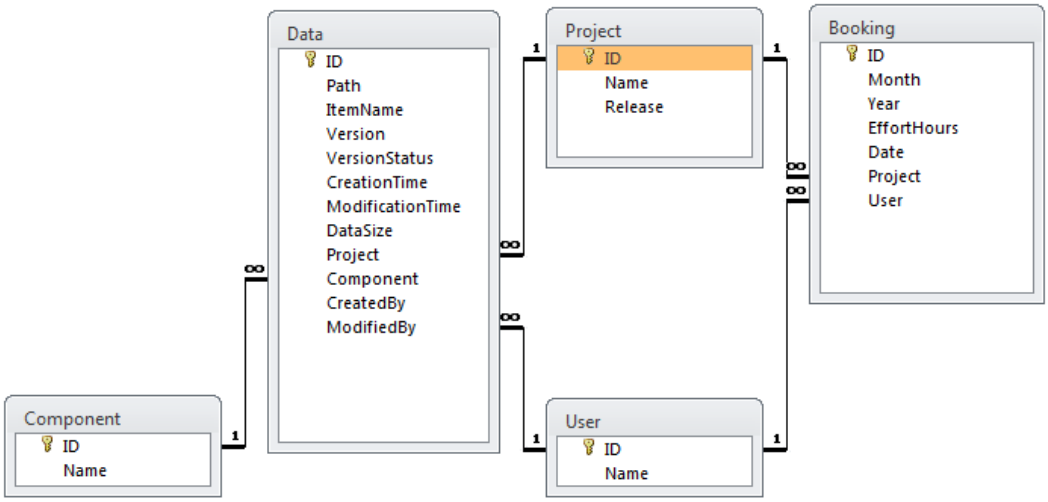

Figure 1: Relational database used for the simulation data

The data used for the simulation (table `Data` in Figure 1) combine with the data from three tables. The `component` table contains the list of distinct software components, which are determined by using the path and filename data. The locations of the component root folders in the source tree are determined by detecting the checked-in versions which have standard extensions in Visual Studio project files, such as .dsp, vcproj, .vcxproj, .csproj, vbproj, .vbp, etc. The `User` table contains all the developers involved in the projects, who both checked in the components and booked their efforts under the relevant project accounts. The `Booking` table holds all the bookings of effort made by the developers during the course of a project. The data in this table combine with the `Project` table and `User` table. Even with the help of the relational database, we still spent a significant amount of time doing data cleansing. In the end, we ended up with 41 projects containing complete and useful data. The 41 projects involved 72 different developers and 607 distinct software components. The average number of components per project was 17, and the average number of developers per project was 5. The average total effort for a project was 784 person hours. With the clean data in the database, we can perform queries to estimate the various parameters we need for the simulation and to retrieve data for validation.

4.2 Simulation Modeling

A project manager at AVL assigns developers to their project based on their availability, skill set and experience. For a project that involves the modification of existing components, a project manager will try to assign developers who are familiar with the components. In the case of developing new components, a project manager will try to choose developers with the required skills and, if possible, experience of similar components. In practice, the ideal developers may be unavailable (e.g. they are working on different projects, on leave, etc.).

The objective of a simulation model is to help a project manager estimate the project completion date for a given set of software components and number of software developers. The boundary of the model reported in this paper covers the design and development phase of a software development project. The model consists of two agents: developer agent and component agent. At any time during the simulation, a developer can be assigned to zero (i.e. unavailable) or more components. A component may have a number of pre-requisite components. A developer cannot work on it until all of its pre-requisite components have been completed. At any time during the simulation, a component can be assigned to zero (i.e. when no developer is available) or more developers (e.g. two or more developers working on the same component). The model is implemented using AnyLogic (www.xjtek.com). AnyLogic represents the behavior of an agent using a state transition diagram. The state diagrams of two agents are shown in Figure 1.

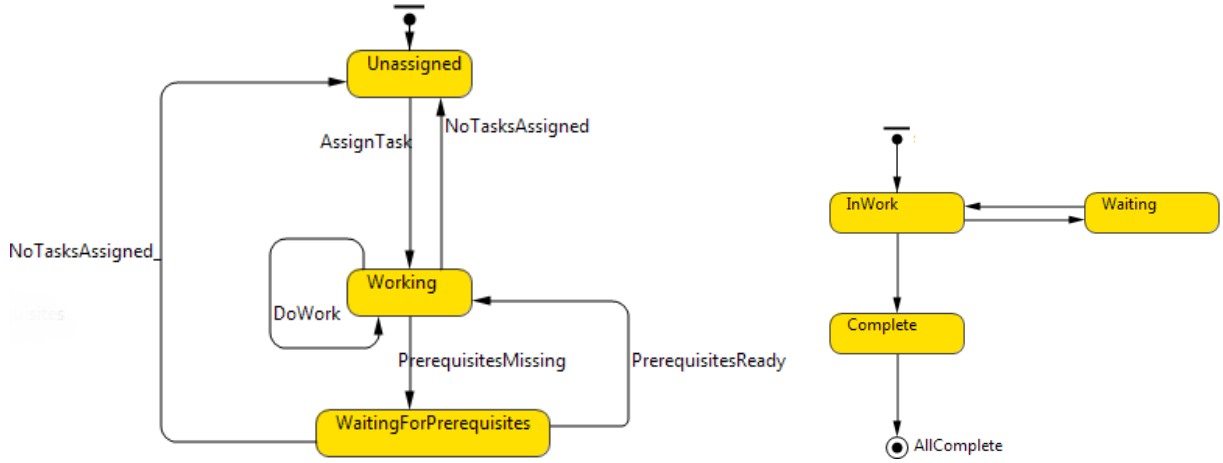

Figure 1: State diagrams representing the behavior of developer agent (left) and component agent (right)

A developer can be in one of three states. An `unassigned` state represents the state where a developer has not been assigned to any component. Once a developer has been assigned to a component, s/he will move to a `working` state. At this state, the developer develops the component based on the given specification document. The time to complete a component is governed by an effort function which will be explained later. The developer may have to wait if the pre-requisite(s) of the component has (have) not been completed by other developer(s). This is represented by the state `WaitingForPrerequisites`.

The component can be in any of three states. Initially, a component has not been assigned to any developer(s). This is represented by the `Waiting` state. Once it has been assigned to one or more developers, its state changes to `InWork`. Finally, when the developer(s) complete(s) the work, the state of the component is changed to `Complete`. A project ends when all components are in the `Complete` state.

We propose an *effort function* to represent the typical pattern of the progress made by a developer during the development of a component assigned to him/her. This function is estimated using changes in the size of all files associated with a component checked in by the developer. This assumes that the development effort of a developer is a function of the change in file sizes over time. The main advantage of this assumption is that, from our experience, the data needed to estimate the effort function are available in many software development companies or departments because they are likely to have been using a version control system where a developer must check in their work before it can be tested and released.

A component C may have one or more files associated with it. In other words, $C(t) = \{F_i^C(t) \mid 1 \le i \le N^C(t)\}$, where C is a software component, F_i^C is the i-th file associated with component C, N^C is the number of files associated with C, and t is the observation time. The size of com-

ponent C at time t, i.e. $S(C(t))$, is $\sum_{i=1}^{N^C(t)} S(F_i^C(t))$, where $S\left(F_i^C(t)\right)$ is the size of F_i^C at time t. If a component is completed in $T = k\,\Delta t$ time units by developer D, then the contribution of developer D to component C in one observation period starting from time $t = i\,\Delta t$ is defined as:

$$v_D^C(i) = \frac{\left|S(C(t+\Delta t)) - S(C(t))\right|}{\Delta t}$$

The proportion of developer D's contribution to component C in the i-th observation period is defined as:

$$e_D^C(i) = \frac{v_D^C(i)}{\sum_{j=0}^{k-1} v_D^C(j)}$$

Hence, for an estimated effort E_0^C given by a project manager at time 0, the total effort delivered by a developer who has been working for τ time units can therefore be represented as an effort function E_D^C:

$$E_D^C(\tau) = E_0^C \int_0^{\tau} e_D^C(t)\,dt$$

At first, we thought the shape of the effort function would depend on the characteristics of the component and developers. However, based on our data, we found that the shapes are relatively similar. Figure 2 shows the average proportion of effort with a 95% confidence interval for each observation period for all distinct combinations of developers and components in our clean data. The number of fixed observation periods used in Figure 2 is 16. The initial peak shows that at the start of an assignment, the developers quickly add the number of lines in their code which could be based on the reuse of existing design patterns or past code. The middle part of the function is relatively stable, with a few spikes, which shows the iterations in the development phases. The final peak in the function shows the stabilization and bug fixing effort made towards the end of an assignment.

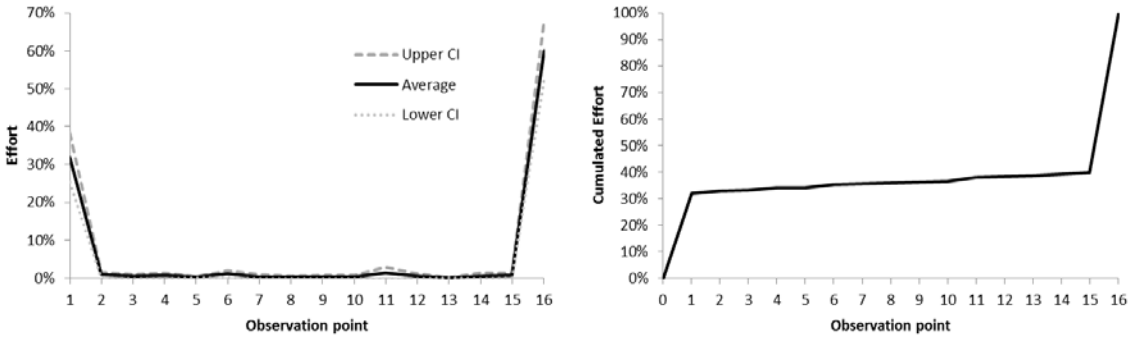

Figure 2: Effort function and cumulative effort function of 72 developers and 607 components (95% CI)

4.3 Validation

We validated the model using various techniques, including white box validation where we evaluated the correctness of the internal working of the model. In this paper, we show black box validation using historical performance data, i.e. project duration. We split the validation into two parts. In the first part, we calibrated each agent in the model with an effort function estimated from his/her individual data. In the second part, we calibrated all agents in the model with the same effort function estimated from the data from all developers (see Figure 2). The results are summarized in Figure 3.

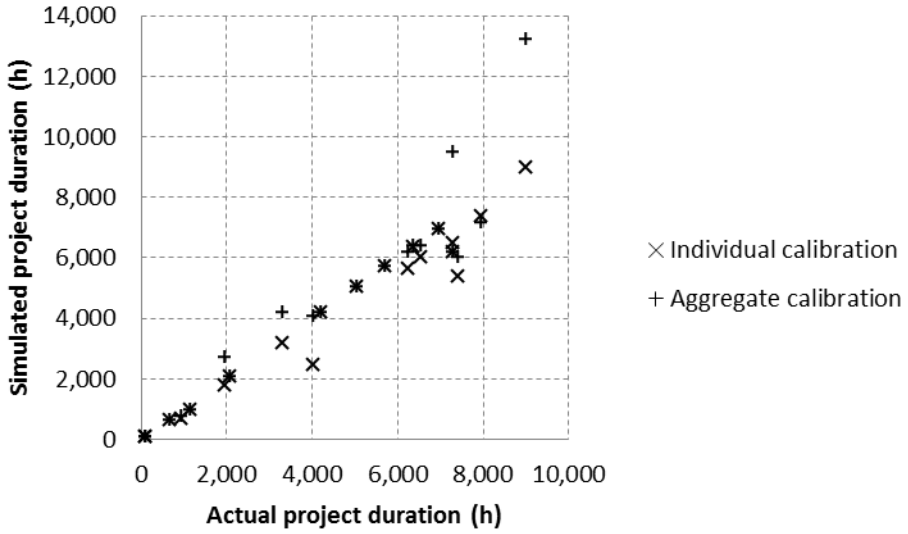

Figure 3: Validation results using individual effort function and aggregate effort function

It shows that our model can estimate project duration relatively well. It also shows that simulation using the aggregate effort function does not perform as well as the simulation done with the individual effort functions for projects of longer duration. This is mainly caused by the accumulation of individual variations. If the developers assigned to a project of very long duration have slightly different behaviors to that of the aggregate effort function, the accumulation may introduce significant bias to the simulated project duration. This implies that for projects of shorter duration, a project manager could simply use the aggregate effort function rather than the more costly-to-implement individual effort functions. This ABS model should help project managers to estimate the duration of future projects, provided that there is no significant change in the behavior of the developers and the complexity of the software components. Otherwise, the model will need to be recalibrated using the same method proposed in this paper.

5 CONCLUSION AND FUTURE WORK

The simulation design presented shows that an agent-based simulation modeling paradigm is a good fit to describe the software development process. We have shown that even with a relatively simple agent-based simulation model, we can estimate project duration well. This shows that an agent-based simulation model does not have to be complex to be useful. In fact, a more complex model would be more demanding in terms of its data requirements. The quality of a model will be largely limited by the availability of good quality behavioral data. It is also relatively easier to communicate a simple agent-based simulation model than a complex one to a project manager. The model presented in this paper was developed, calibrated and validated using data commonly available from a version control system. This makes our approach more practical. Finally, central to this paper is the proposed effort function to estimate the behav-

ior of a developer for a given task in a project. The project managers at AVL can understand and accept the concept of effort function. This further increases the buy-in from potential users.

The model presented in this paper covers the design and development phase in a software development project. It will be interesting to see how this model can be extended to cover all phases in a software development project. An extended model will include more agents, such as testers, and hence more behaviors and interactions. The main challenge will come from finding the right balance between model clarity (so that it can be communicated easily to a project manager) and level of model detail. From our experience, to find readily available data to calibrate an extended model could also be challenging.

ACKNOWLEDGMENT

We would like to thank the Director of ITS Global Research & Technology Management at AVL, Dr. Michael Paulweber, for his kind support and sponsorship of this work.

REFERENCES

Abdel-Hamid, T., and S. Madnick. 1991. *Software Project Dynamics: An Integrated Approach*. Upper Saddle River, New Jersey: Prentice-Hall, Inc.

Agarwal, R. 2007. "A Flexible Model for Multi-Agent Based Simulation of Software Development Process." Ph.D. Thesis, Department of Computer Science and Software Engineering, Auburn University, Auburn, Alabama. http://etd.auburn.edu/etd/bitstream/handle/10-415/190/Agarwal_Ravikant_38.pdf (Accessed April 3, 2012).

Agarwal, R., and D. Umphress. 2010. "A Flexible Model for Simulation of Software Development Process." In *Proceedings of the 48th Annual Southeast Regional Conference (ACM SE'10)*, 4 pages. New York, NY: ACM.

Cherif, R. 2008. "Software Process Simulation Modelling: A Multi Agent-Based Simulation Approach." M.Sc. dissertation, Blekinge Institute of Technology, Karlskrona, Sweden. http://www.bth.se/fou/cuppsats.nsf/all/944f942929cd8e9fc12573ed0069b353/$file/Software_Process_Simulation_Modelling_A_MABS_Approach.pdf (Accessed: April 3, 2012).

Dong, S., and B. Hu. 2008. "Multi-Agent Based Simulation of Team Effectiveness in Team's Task Process: A Member-Task Interaction Perspective." *International Journal of Simulation and Process Modelling* 4 (1):54–68.

IBM. 2007. "IBM Rational Unified Process". ftp://public.dhe.ibm.com/software/rational/web/datasheets/RUP_DS.pdf (Accessed April 3, 2012).

Johnson, P. M., H. Kou, J. Agustin, C. Chan, C. Moore, J. Miglani, S. Zhen, and W. E. J. Doane. 2003. "Beyond the Personal Software Process: Metrics Collection and Analysis for the Differently Disciplined." In *Proceedings of the 25th International Conference on Software Engineering*, 641–646. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Joslin D., and W. Poole. 2005. "Agent-Based Simulation for Software Project Planning." In *Proceedings of the 37th Winter Simulation Conference*, Edited by M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, 1059-1066. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Kellner, M. I., R. J. Madachy, and D. M. Raffo. 1999. "Software Process Simulation Modeling: Why? What? How?" *Journal of Systems and Software* 46 (2-3):91–105.

Raffo, D. M., and W. Wakeland. 2008. "Moving Up the CMMI Capability and Maturity Levels Using Simulation." Carnegie Mellon SEI http://www.sei.cmu.edu/reports/08tr002.pdf (Accessed: April 3, 2012).

Smith, N., A. Capiluppi, and J. Fernández-Ramil. 2006. "Agent-Based Simulation of Open Source Evolution." *Software Process: Improvement and Practice* 11(4):423–434.

Yilmaz, L., and J. Phillips. 2007. "The Impact of Turbulence on the Effectiveness and Efficiency of Software Development Teams in Small Organizations." *Software Process: Improvement and Practice* 12 (3):247–265.

Wickenberg, T., and P. Davidsson. 2003. "On Multi Agent Based Simulation of Software Development Processes." In *Multi-Agent-Based Simulation II*, Edited by J. Simão Sichman, F. Bousquet, and P. Davidsson, vol. 2581, 72–77, Berlin: Springer Berlin-Heidelberg.

Zhang, H., B. Kitchenham, and D. Pfahl. 2008. "Reflections on 10 Years of Software Process Simulation Modeling: A Systematic Review." In *Making Globally Distributed Software Development a Success Story*, Edited by Q. Wang, D. Pfahl, and D. Raffo, vol. 5007, 345–356. Berlin: Springer Berlin-Heidelberg.

Zhang, H., B. Kitchenham, and D. Pfahl. 2010. "Software Process Simulation Modeling: An Extended Systematic Review". In *New Modeling Concepts for Today's Software Processes*, Edited by J. Münch, Y. Yang, and W. Schäfer, vol. 6195, 309–320. Berlin: Springer Berlin-Heidelberg.

Zhang, H., R. Jeffery, D. Houston, L. Huang, and L. Zhu. 2011. "Impact of Process Simulation on Software Practice: An Initial Report." In *Proceedings of the 33rd international conference on Software engineering*, 1046–1056. New York, NY: ACM.

AUTHOR BIOGRAPHIES

BOJAN SPASIC is a project leader for software development projects in the Instrumentation and Test Systems business area at AVL, world's largest privately owned and independent company for the development of powertrain systems with internal combustion engines, as well as instrumentation and test systems. He has an MSc in Information Systems Management from the University of Liverpool. He has 16 years of professional experience in software development, including six years in managing software development projects. His email address is bojan.spasic@avl.com.

BHAKTI STEPHAN ONGGO is a lecturer in Business Process Modeling and Simulation at the Department of Management Science at the Lancaster University Management School. He completed his PhD in Computer Science from the National University of Singapore and his MSc in Management Science from the Lancaster University. His research interests are in the areas of simulation methodology (modeling paradigms and conceptual modeling), simulation technology (parallel and distributed simulation) and business process modeling and simulation applications. His email address is s.onggo@lancaster.ac.uk.