ProSim'05

The 6th International Workshop on Software Process Simulation and Modeling

14-15 May 2005

Saint Louis, Missouri, USA

ICSE 2005 Co-located Event

Editors: **Dietmar Pfahl,**
David Raffo,
Ioana Rus,
Paul Wernick

Printed by Fraunhofer IRB, Stuttgart, Germany, 2005, ISBN 3-8167-6761-3

Table of Contents

ProSim'05 - The 6th International Workshop on Software Process Simulation and Modeling

Part 4: Position Papers

Message from the Steering Committee

We would like to welcome you to St. Louis, USA, for the 6h International Workshop on Software Process Simulation and Modeling (ProSim'05).

Since 1998 ProSim has been a successful international workshop that has show-cased the leading research in the Software Process Simulation and Modeling domain. Moreover, it has become one of the regular international meetings of the Software Process community. Participants have come from Europe, Asia, South America, North America, Africa, and Australia/New Zealand. With respect to structure, the workshop is a combination of keynote and paper presentations, themed sessions, and panel discussions.

The goal of this workshop is to bring together academics and practitioners interested in the area of software process modeling and simulation and in important industrial issues related to cost estimation and business process design. ProSim'05 continues the tradition set in previous workshops of serving as an international forum for presenting current research themes and applications, and for discussing various approaches to discover underlying similarities at both the applied and theoretical levels. In particular, this workshop will solicit research dealing with both the application of software process simulation research in addressing real-world problems, as well as advances being made which provide the foundation for Software Process and Software Process Simulation Modeling in the future.

We would like to thank the members of the ProSim'05 program committee, as well as the session chairs and all who have helped to set up this conference and made the co-location with ICSE'05 possible.

Dietmar Pfahl
Fraunhofer IESE, Germany
pfahl@iese.fraunhofer.de

Ioana Rus
Fraunhofer Center, USA
irus@fc-md.umd.edu

David Raffo
Portland State University, USA
davidr@sba.pdx.edu

Paul Wernick
University of Hertfordshire, UK
P.D.Wernick@herts.ac.uk

Workshop Organizers

Steering Committee

Dietmar Pfahl
Fraunhofer IESE, Germany

David Raffo
Portland State University, USA

Ioana Rus
Fraunhofer Center, USA

Paul Wernick
University of Hertfordshire, UK

Web-Page Design

Siri-on Setamanit
Portland State University, USA

Proceedings Design

Niniek Angkasaputra and Stephan Thiel
Fraunhofer IESE, Germany

Program Committee

Thomas Birkhoelzer, *University of Applied Sciences Konstanz, Germany*

James Collofello, *Arizona State University, USA*

Paolo Donzelli, *University of Maryland, USA*

Volker Gruhn, *University of Leipzig, Germany*

Ross Jeffery, *UNSW and NICTA, Australia*

Dan Houston, *Honeywell, USA*

Marc Kellner, *Software Engineering Institute, CMU, USA*

Marek Leszak, *Lucent Technologies Bell Labs, Germany*

Ray Madachy, *University of Southern California, Los Angeles, USA*

Bob Martin, *Software Management Consulting, USA*

Jürgen Münch, *Fraunhofer IESE, Germany*

Leon Osterweil, *University of Massachusetts, USA*

Dewayne Perry, *University of Texas, Austin, USA*

Dietmar Pfahl, *Fraunhofer IESE, Germany*

Antony Powell, *University of York, UK*

David Raffo, *Portland State University, USA*

Juan F. Ramil, *The Open University, UK*

Günther Ruhe, *University of Calgary, Canada*

Mercedes Ruiz Carreira, *Escuela Superior de Ingenieria, Cadiz, Spain*

Ioana Rus, *Fraunhofer Center Maryland, USA*

Walt Scacchi, *University of California, Irvine, USA*

Thomas Thelin, *Lund University, Sweden*

Paul Wernick, *University of Hertfordshire, UK*

Part 1

Keynotes

Keynote 1:

"Estimating the Risk of Releasing Software"
by Brendan Murphy, Microsoft Research, UK

Abstract:
In the ideal world all commercial software would be fault free but, unfortunately, there is no known process and or tools that can guarantee such software. Therefore to improve the overall quality of Windows software a number of studies have been performed to understand the relationship between software development attributes (such as size, complexity, churn) and subsequent failures affecting end customers. This talk will describe the results of some of these studies, the process used to develop Windows 2003 operating system and how the results of the studies are being fed into models to predict the risk associated with releasing versions of the Windows operating System.

Keynote 2:

"Open Market Software Development"
by David M. Weiss, Avaya Labs, USA

Abstract:
Two critical issues in a software development organization are how work is assigned to developers and how developers are compensated. Although these may sound like organizational issues, they are closely linked to technical issues, especially architectural issues. A development process such as software product line engineering, which often focuses on a common architecture for a product line, provides an opportunity to change the usual ways that work is assigned and developers are compensated. Open market software development is a proposal for making such a change, allowing developers more freedom to choose their work assignments and compensating them based on the value of their work.

Keynote 3:

"Predicting Value from Design"
by Mary Shaw, Carnegie Mellon University, Pittsburgh, USA

Abstract:
Early design decisions in software projects profoundly affect both the properties and the costs of the eventual implementation. It is much easier and cheaper to change these decisions during design than after implementation has yielded running code. Improvements in our ability to predict properties of an implementation without actually inspecting the code would enable software designers to better understand the consequences of early decisions and would facilitate comparison of design alternatives to a degree not currently possible. This talk will discuss some code-free predictive evaluation techniques and the challenges of harnessing them to provide a unified framework for reasoning about the overall value that should arise from a design.

Part 2

Research Papers

Section I

Process Modeling – Focus on Methodology

Model Support for Simulation-Based Training Games: From Behavioral Modeling to User Interactions

Gustavo Olanda Veronese[1], Marcio Oliveira Barros[2], Cláudia Maria Lima Werner[1]

[1] COPPE – UFRJ
Systems and Computer Engineering Program
Caixa Postal 68511 - CEP 21945-970
Rio de Janeiro – RJ – Brazil
{veronese, werner}@cos.ufrj.br

[2] DIA – UNIRIO
Av. Pasteur 458, Urca
Rio de Janeiro – RJ – Brazil
Voice: 55 21 2244-5613
marcio.barros@uniriotec.br

Abstract

Current trends in education point to an increasing use of simulation-based environments for learning purposes. However, the industry still lacks well-documented techniques, models, and processes to organize the development of simulation-based educational games. In this paper, we propose a framework to support the development of educational games based on simulation models. The framework is composed by a lightweight process and a set of models that emphasize (i) the separation of activities regarding distinct aspects of a game; and (ii) the reuse of artifacts that were built to represent such aspects throughout game development. A software project management training game was modeled as a case study of the proposed approach.

1. Motivation

Simulation-based games are drawing the attention of many researchers [3][4][5], who are analyzing their usage as a supporting technology for conventional methods of adult training, among other applications. It has been observed that adults tend to learn better if they can experiment, make mistakes, and apply the theories that they have learned in practical situations. Simulation-based instructional environments allow trainees to play an active role in the learning process, providing a virtual representation for situations with which the trainees can interact and apply their knowledge without the risks associated to the real world [1].

Even though many researchers agree on the relevance of simulation-based games to educational purposes, there is still a lack of well-documented processes and integrated tools to support the organization of development activities to build such games. Moreover, current game development tools, such as game templates and engines, are not easily integrated to simulation models.

Ideally, simulation-based educational games should be easily adapted to distinct training demands. From a software designer perspective, the development of adaptable training games involves the identification of common aspects that constitute this kind of software, the formal representation of such aspects, and the composition of games based on these representations (preferably reused from formerly developed games).

Therefore, in order to reduce the effort invested in the development of simulation-based games, there is a need to provide formal representations for common aspects involved in game development and a composition technique to build games from these representations.

In this paper, we present a framework that supports the development of simulation-based training games. A lightweight development process, a set of models, and a game composition strategy compose the framework. Currently, the proposed models focus on the development of single-user, two-dimensional games, where the trainee interacts with model elements through a graphical interface. To analyze the usefulness of the proposed framework, a training game in the field of software project management was developed as a case study. The game presents a project to the player, where he must execute management tasks, such as controlling project execution, staffing, and planning.

This paper is organized in three sections. The first section comprises this motivation. Section 2 presents the fundamentals of the proposed

approach. Section 3 describes the software project management game that was modeled as case study to evaluate the proposed framework. Finally, section 4 presents conclusions and future perspectives of this work.

2. Model Support for Simulation-based Training Games

In a training game, the characters and objects with which the trainees can interact represent concepts of the knowledge area that is under exploration in the educational context. These elements can have their behavior controlled by simulation. By behavior, we mean how these elements respond to changes in the modeled system and to user interactions. Educational games usually use graphical effects to enrich their interactivity and fantasy, trying to capture user attention and motivation. From this view, we identify three main aspects that should be covered while developing a simulation-based game: a simulation component, a story that contextualizes what is being learned, and interaction elements.

In this paper, we propose a model-driven game development framework that divides a training game project into these three major aspects. A distinct model maps each of these aspects:

(i) a *behavioral model* describes how the elements within a domain behave over time, according to user interactions;

(ii) a *story model* maps the mathematical or logical formulations presented in the behavioral model to elements pertaining to the game domain, such as objects and characters;

(iii) a *graphical model* depicts the elements composing the story model, presenting them to the user and controlling user interaction with the story model.

The framework's game composition strategy receives one model for each of these aspects and integrates their references. The composition strategy resembles a layered architecture, where the behavioral model simulator represents the core layer. A second layer manages the story model, commanding simulation steps upon the behavioral model, capturing results calculated by the simulator, and mapping these results to domain level, user recognizable states for the characters and objects that compose the story. Finally, the third and upper layer handles user interactions, translating user actions to story level events, capturing story elements and presenting them to the user. Therefore, the development of a simulation-based educational game requires building the behavioral, story and graphical models.

The skills required to build the behavioral model (domain knowledge, mathematical modeling, logical modeling, among others) seem very distinct from the skills that are needed to develop the graphical model (visualization, spatial organization, drawing, etc). So, we assume that several persons will be involved in the development of an educational game. The separation of distinct game aspects in different models supports work distribution among the game development team. To coordinate the work performed by these distinct groups, we propose a lightweight game development process.

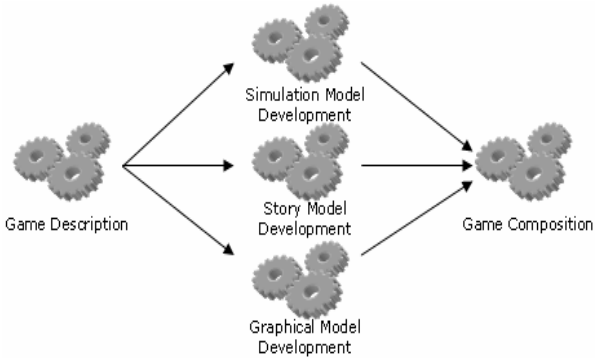


Figure 1 – Overview of the proposed game development process

Five steps compose the proposed process, as presented in Figure 1. The first one corresponds to the elaboration of a short textual description for the desired game, describing player's goals, domain elements involved in the game (characters and objects), environments where the game story takes place, and possible states that can be assumed by game elements.

The three central activities of the proposed process have the objective of designing the aspects into which the game was divided: (i) simulation definitions; (ii) story definitions; and (iii) graphical definitions. First, the development team describes the simulation model. Next, they build the story model. Finally, the team builds the graphical model. Such models are addressed in detail in the following subsections.

Since distinct persons tend to participate in the development of each model, these activities can run in parallel. However, this requires the definition of interfaces among the model layers, to reduce integration effort. The game textual description supports these definitions, providing a common view of the desired game to be shared among developers.

The last process step regards the composition strategy. In this activity, the models built by the preceding activities are integrated and executed. In the execution phase, two roles take place: the trainee (game player), and the facilitator, whose main function is to elucidate trainee's doubts and questions.

In the following subsections, we address each of the models used in the framework in detail. To support the presentation, we discuss the models in the light of our project management educational game case study.

2.1 The Simulation Model

The simulation model is the core of the game. It defines the elements that take part in the problem domain addressed by the game and the relevant relationships among these elements. The model describes, through mathematical formulations, the rules that drive each domain element behavior. The simulation model is described in the System Dynamics Metamodel notation [3], an extension of the System Dynamics modeling language in which models are described in a high level representation, closer to object oriented concepts.

This metamodel allows one to design domain elements as classes, each composed by properties and behavior. Properties are relevant numeric data about the class, while class behavior is denoted by a set of equations describing rules of how class instances react to changes in the model. An instance is a specific occurrence of a class in a model. Each class instance can have different values for the properties defined in its class. Class level relationship constructs define how underlying instances are linked to each other when the model is under execution. Although we have defined instances at this point, the instances involved in the game are handled in the story model.

In our project management case study, problem domain elements include the project, its activities, and the developers composing the project team. Activities are classified as analysis, design, coding, tests, and inspection activities. Relationships among these elements include the developer assigned to accomplish an activity, the activities that compose a project, and the activities that must be completed before start of another activity.

Figure 2 depicts the Developer and Activity classes defined in a graphical notation of the simulation model. The notation resembles UML diagrams, though simulation model diagrams present system dynamics related elements, such as stocks and rates, instead of attributes and methods. Each three-sectioned rectangle represents a class, whose name is presented in the upper section. Class properties are positioned in its middle section, while behavior is presented in the lower section.

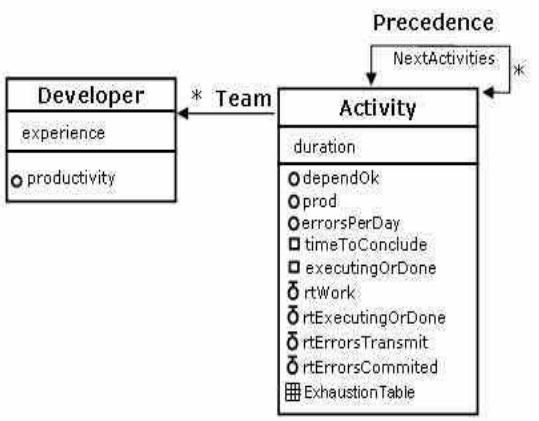


Figure 2 – Simulation Model Diagram

The directional association between the Activity and Developer classes allows a developer to be assigned to accomplish an activity. The relationship only allows instance from these classes to be linked to each other, but behavior equations can use these links to represent the behavior of developers executing an activity. The auto-relationship named as Precedence defines the execution order of project activities, since it describes which activities precede each activity.

2.2 The Story Model

The story model is built atop the simulation model. Its major objective is to map the abstract equations that compose the simulation model into real world concepts that are easily recognizable by the trainee. To accomplish this mapping, the story model assigns descriptive attributes to each simulation model element, such as the name and curriculum of a developer, the details on how an activity should be developed, among others. Such attributes are not relevant for the simulation, that is, they do not impact on the model behavior1, but they are helpful to facilitate the interactions between the trainee and the simulation model. Instead of handling with equations, the trainee interacts with elements that can be observed in the problem domain (a developer with a name, an activity with procedures for its execution, and so on).

In addition, the interactive nature of games required some changes on how the simulation is conducted by the System Dynamics Metamodel simulator. In the original simulator, interactions between the modeler and the model are limited to

1 In case they could affect model behavior, such attributes should be addressed in the simulation model and would not be considered descriptive attributes.

an initial setup. Once model parameters are set, the remaining simulation steps are automatically run, without further intervention from the user. However, trainees in educational games are frequently adjusting the underlying model by making decisions that concern game elements. These decisions, such as changing the developer assigned to an activity or the number of hours that a developer is asked to work per day, change model values and structure, and thus have to be accounted for in the remaining simulation cycles. To support structural changes in the relationships among model elements after the simulation cycles have started, we have changed the simulator implementation.

The story model is responsible for mapping user interactions with story elements into structural changes in the simulation model. The model associates a set of actions with each domain element. User interactions can trigger these actions. Each action is described as a sequence of primitive structural operations that affect the underlying behavioral model. Such operations include changing the value of a property, creating a new domain element (such as a developer or an activity), removing an element from the model, breaking a relationship between two elements, or creating a new relationship among such elements. The new simulator handles these operations by changing the simulation model structure, storing the values calculated in previous simulation cycles according to the former structure, and calculating future cycles with the new one.

A state machine controls the actions that can be enacted upon a domain element by the trainee in a given moment. This state machine is defined in the story model and presents the relevant states for each domain element that can be perceived by the trainee. Each state is associated with a set of actions, and state transitions are triggered by changes in the simulation model variables. Transitions are defined as boolean expressions based on the value of simulation variables.

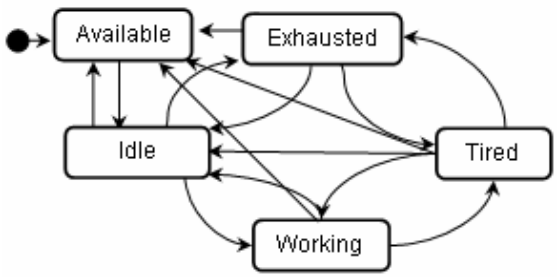


Figure 3 – Developers' state machine

Figure 3 shows an example of a state machine designed to control developer interactions in the case study. It dictates how each developer will react to changes in the simulation model in consequence of user interactions. When the project starts, developers are available to execute any activity. They automatically pass to the "Idle" state when hired. After being assigned to some activity under execution, they pass to the "Working" state. Depending on the developer's exhaustion level, its state can alternate among "Working", "Tired" or "Exhausted". Developers only return to the "Available" state if they are fired. So, three information given by simulation variables determine a developer's state: allocation to the project, exhaustion level, and association to one or more activities under execution. Other game elements, such as project activities, have distinct and independent state machines.

After defining new structures, like action and attributes, to be embedded in simulation classes, and state machines for each class, there is a setup step where the story modeler should initialize all instances in the game. For example, instances of "Activity" class are named "Analysis", "Design", "Inspection", "Coding", and "Testing", all of them with distinct values for the "function points" property.

2.3 The Graphical Model

The graphical model contains visual representations (images and animations) for story elements. Decorative objects (such as desks, chairs, and pictures) are added to the game. Story level state machines are associated to presentation level state machines, whose goal is to reflect visually the changes occurred in a story element. So, story-level state transitions trigger actions in the graphical layer, such as moving or disposing an image. By using this strategy, it is possible, for instance, to distinguish a tired developer from a developer that is not exhausted: sometimes, the tired developer will be sleeping on the desk, asking for coffee, and so on.

Figure 4 shows all possible graphical states that a developer can assume, when he/she is in the working state. Some graphical states can be represented by an animation (as listening to music) and some can be static images (as thinking). The switch among graphical states occurs according to probability distribution stated in the model.

The graphical model also defines interaction environments (such as offices, laboratories, and rooms), which are linked by navigation paths associated to story objects. Finally, the graphical model maps user level interactions, such as mouse movements or the activation of mouse buttons upon the visual representation of a story element, into story level actions. This completes the cascade

of user interactions, from mouse actions to changes in the behavioral model structure, through the three-layered game architecture.

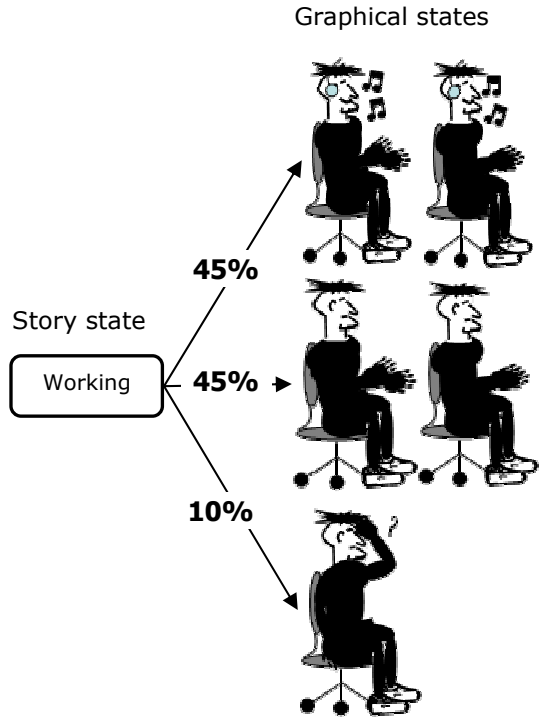


Figure 4 – Graphical states for the state "Working"

3. A Project Management Training Game

To evaluate the usefulness of the framework and its models, we have developed a project management game using the proposed approach. The game starts with a textual description of an academic system to be developed in a given time and with limited budget. The corresponding simulation model is a representation of the real system, developed and used by a Brazilian graduation program. In the proposed problem, the manager should use the waterfall process model.

After an initial project presentation, the manager can see a human resources room, where developers are available to integrate the staff. By clicking on a developer, the player can evaluate if it is worth to hire him, based on characteristics including skills on some project activities and hourly cost.

Figure 5 shows a screenshot extracted during a game session. It represents the laboratory where developers work. The player can interact with developers and activities (the later are shown in the board placed on the wall), change the current environment (by clicking in the door), and execute generic commands in a control panel (on the bottom of the screen). Computers, chairs, desks, printers, and telephones are just decorative objects.

In the illustrated scenario, the player is changing the developer assigned to an activity. Activity execution can be analyzed by looking at the board. In this project, the two parallel threads of activity execution represent two distinct use cases under development. Concluded activities are marked with "C" (analysis and design for both use cases). Balls with triangles indicate on-going activities. Activities marked with the encircled square are disabled (in this case, inspections). Paused activities, represented by parallel rectangles inside a circle, are waiting for the conclusion of precedent tasks.

Some relevant variables are continuously displayed on the control panel during the game session. The right-hand area of the control panel shows the elapsed time since the project started and the budget consumed until this moment. These variables are chosen in the graphical model and should help the player in taking decisions. The game ends when either the project is concluded successfully or allocated time/budget are consumed (implying project failure).

Although the simulation model was reused from another application, we had to adapt it to support the story-level state machines that were required by the *Developer* domain element. Basically, some transitions in the developer's state machine needed to query information that was not exposed by its related domain model class. For instance, we included a boolean variable to inform whether the developer was allocated to some project activity or not. This variable allowed us to create the human resources office, were available developers waiting to be hired are presented during the game.

Moreover, we realized that more states, either in the story level or in the interface would improve user awareness about what is happening during the game. So, the game designer has to make a tradeoff between awareness and simulation complexity, since each new state in the story or presentation layers requires several transitions to be included and mapped into the simulation model.

While developing the case study, certain difficulties were introduced because the proposed models are curretly developed in common text editors: simulation models are written according to the metamodel notation [2], and the story and graphical models are based on a particular XML schema. Some defects could only found during execution. Most of these defects were traced to syntax problems that could be found during model development, thus informing the game designer earlier than the test phase. We plan to prevent such errors by developing specialized model editors with rule verifiers.

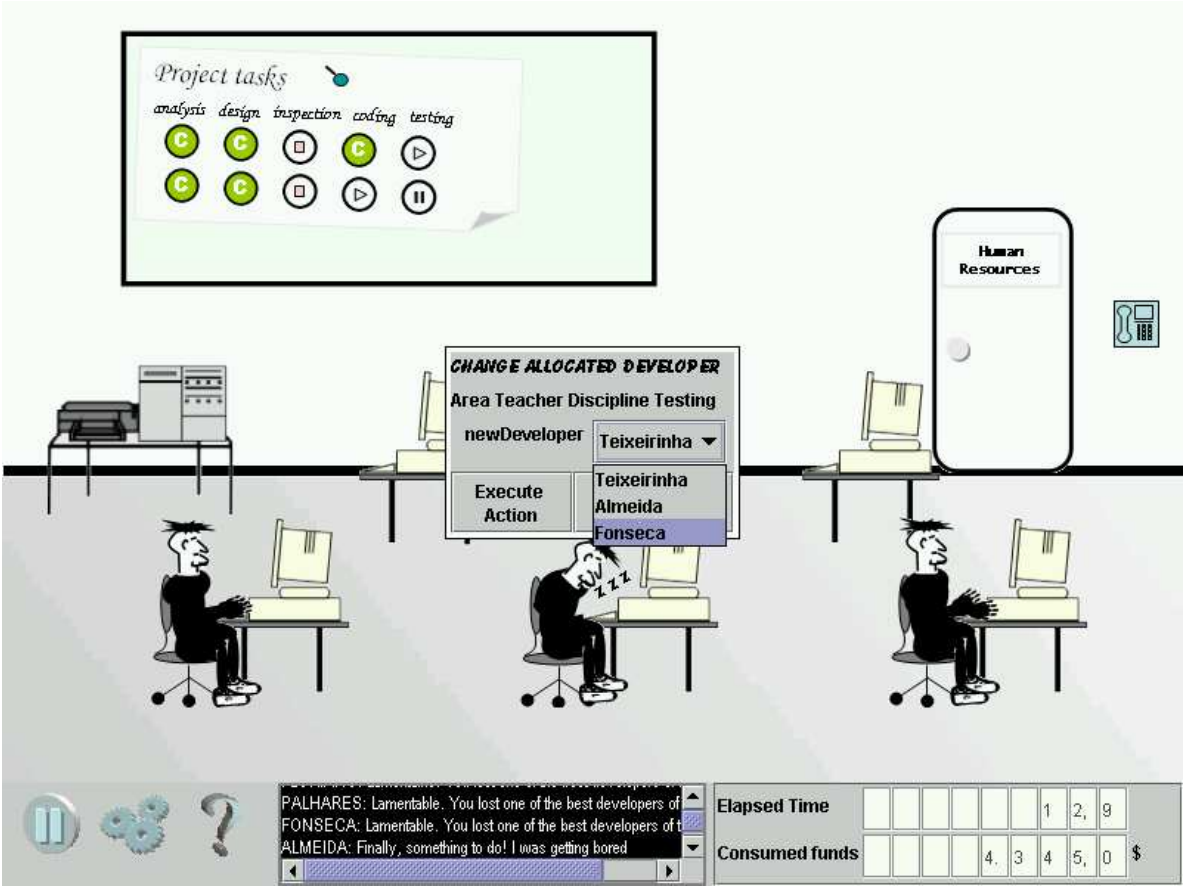


Figure 5 – Project Management Training Game: Laboratory Environment

4. Conclusions and Future Work

In this paper, we presented an approach to organize the development of simulation-based educational games. Such approach is based on a lightweight development process, which is centered in three models addressing distinct game aspects (behavior, story, and presentation). The separation of game concerns into these models is supposed to allow an easier game development. The evolution of one aspect does not directly affect the other two. For example, designers can change characters configuration and problem modeling by changing story and graphical model (if necessary). No work ought to be done in the simulation model.

We believe that one of the main benefits of the approach is that it allows designers to focus on domain aspects, saving development time, which otherwise would be wasted in specific programming level problems, such as event handling, graphics rendering, and others. The game execution engine is responsible for controlling all these common aspects of simulation games. Another benefit is that the proposed models are not dependent of a specific simulation model. Thus, several behavioral models for the same domain may share the same story model. This allows the preparation of several instructional lessons, each with a proper model addressing a distinct issue, using the same story and graphical layers.

Although the approach has been tested in a specific game, an effort should be made in performing experiments to evaluate its feasibility in other problems. Also, future research should focus in developing mechanisms for the semi-automated analyses of game events occurred during training sessions. These analyses should permit assessing the efficacy of learning experiences based on this kind of simulation games. They should also address the issues that were not fully understood by the trainee for more focused lessons.

References

[1] Ahdell, R. e Andresen, G. (2001). Games and simulation in workplace e-learning. Master's thesis, Norwegian University of Science and Technology, Norway.

[2] Barros, Márcio O.; Werner, Cláudia M.L.; Travassos, Guilherme H. (2001) "From Models to Metamodels: Organizing and Reusing Domain Knowledge in System Dynamics Model Development", in *Proceedings of the 19th International Conference of the System Dynamics Society*, Atlanta, USA

[3] Groβler, A. (2000). Methodological issues of using business simulators in teaching and research, in *Proceedings of 18th International conference of The System Dynamics Society*, 18, Bergen, Norway.

[4] Oh, E., Baker, A., and van der Hoek, A. (2004). Teaching software engineering using simulation games, in *Proceedings of the 2004 International Conference on Simulation in Education*, San Diego, California.

[5] Spector, J. M. (2000). System dynamics and interactive learning environments: Lessons learned and implications for the future. Simulation & Gaming, 31(4):509-516.

Self-Organized Development in Libre Software: a Model based on the Stigmergy Concept

Gregorio Robles
Universidad Rey Juan Carlos
grex@gsyc.escet.urjc.es

Juan Julian Merelo
Universidad de Granada
jmerelo@geneura.ugr.es

Jesus M. Gonzalez-Barahona
Universidad Rey Juan Carlos
jgb@gsyc.escet.urjc.es

Abstract— **Libre (free, open source) software projects are lately getting increasing attention from the research community; for instance, several studies have focused on the inner working of some successful projects. However, there is still little emphasis on trying to explain the landscape of libre software development at large, maybe due to the distribution of developers, to the (in many cases) non-compulsory nature of their relationships, and to the extreme importance of motivation to attract resources to a project. In this paper we model the relationships among developers (with each other and the projects they decide to put work in) with the behavior of some social insects performing large-scale works. Specifically, we apply the concept of stigmergy, which considers that communication (by means of stimulus) does not happen directly among entities (in our case developers), but through changes in the environment. Stigmergy makes an autocatalytic reaction, of the same kind that the one observed in bazaar-like self-organized libre software projects, possible. We will build a model based upon these ideas, test it against quantitative data and results from previous research, and provide results of a simulation. Our conclusion is that the libre software development can indeed be modeled as a stigmergic phenomenon, in terms of allocation of developers to projects, and in the further evolution of those projects. An important consequence of this fact is that the individual productivity of developers would be not as important as the total production of a community. This would mean that the exploitation of stigmergic mechanisms would be more efficient for increasing the output of a project than actions oriented towards increasing productivity of individuals.**

Keywords: Process modeling and simulation, libre (free/open) source software, self-organization, software evolution

I. Introduction

One of the most surprising characteristics of libre software development is that it does not seem to obey many rules of the 'classical' software engineering development process. There are no (or few) pre-defined requirements, no detailed design, and a lack of inter-process documentation [1]. In addition, libre software projects are not organized in a clear, predefined hierarchical structure, where a central authority shows the way to go. But despite all of that, they are capable of delivering useful, mature and (in many cases) high quality pieces of software. This can only be explained by assuming they feature self-regulation and self-organization.

Most of these circumstances have been known for many years [2], and there has been plenty of research activity around those topics [3]–[6]. However, there is still no good model on how libre software is produced as a whole. Maybe because the inherent difficulties of dealing with many projects, each with different contexts, research effort has been focused on single, usually successful, libre software projects.

In this paper, we are interested in the complete landscape of libre software development. We propose a model based on stigmergy, a concept that helps to explain how some social insects perform large-scale works. Stigmergy assumes that communication between individuals happens through stimuli caused by changes in the environment, and not by directly interchanging information. Stigmergy explains how autocatalytic reactions are possible: the same kind of behavior observed in bazaar-driven self-organized libre software projects.

Based on this stigmergy concept, we have designed a model for studying the evolution of the libre software landscape, and of the projects in it. The model deals with how developers are allocated to projects (or how developers decide to join projects), and how that impacts on the relative evolution of those projects. We have also implemented a system for the simulation of the model (which has been calibrated using data from previous studies), and verified its output comparing its results to the results of the research on real libre software projects.

This paper is, then, a first step towards understanding the social and computer-mediated interactions that yield, as a final product, a libre software project, and, as such, it can be used to improve those interactions in order to produce software products more efficiently, even in a non-libre software environment.

The structure of this paper is as follows: Next, social insects and the concept of stigmergy are presented (section II. After that, the application of stigmergy to libre software development is discussed in section III, including the description of a model for libre software development; the next section IV indicates how the model has been verified. The next section (V) discusses the model, its implications and its limitations. To finish the main part of the paper, some conclusions are presented in section VI. After that there are three appendices, detailing the conceptual model, the implementation of a simulation for it, and the verification of the model comparing the results of simulations with real data about projects.

II. Self-organization through stigmergy

In the late 1950s the French biologist P.P. Grassé realized, while studying the construction of termites nests, that

The work of Gregorio Robles and Jesus M. Gonzalez-Barahona has been funded in part by the European Commission, under the CALIBRE CA, IST program, contract number 004337, by the Universidad Rey Juan Carlos under project PPR-2004-42 and by the Spanish CICyT under project TIN2004-07296. We thank the anonymous reviewers of the ProSim 2005 workshop for their comments and suggestions.

some behaviors which lead to collective coordination were consequence of the effects produced in the local environment by previous behaviors (usually of other termites). He called this phenomenon stigmergy[1] [7].

Grassé observed that when termites build their nest, they start randomly, without any coordination. Once a certain point of activity is reached in an areal, it becomes a significant stimulus for other termites which then start to collaborate in that same area, leading to the construction of the nest.

Stigmergy is observed mainly in social insects, such as termites, ants and some kinds of spiders. Their activity does not depend on the direct interactions with other insect-workers, but on the structure of the environment. Individual behavior is controlled and guided by previous work, i.e. the changes in the environment have a direct impact on the self-organization and coordination of the colony. An insect creates, with its activity, a structure which stimulates other members of the colony, causing them to perform other specific activities.

One of the ways stigmergy is observed in social insects is by means of chemical marking, depositing pheromones in specific places (for instance where food is found). Several deposits in the same place have an additive effect, causing in the end an autocatalytic reaction[2]. Stigmergy is, therefore, a coordination mechanism, characterized by the lack of *a priori* planning and explicit or direct communication between entities. Information exchange is done through changes in the environment, which usually have only local effects, and can therefore exert influence only nearby where they were produced.

Stigmergy has already been recognized as a method for studying cognitive processes, such as software development by Susi [8], who compared different theoretical frameworks for artifact-mediated collaborative activity. In the case of software development, artifacts are on-line communication media (wikis, mailing lists, and the source code itself).

In our study we apply the stigmergy concept to libre software development. Although in general ant algorithms have been used for optimization in several contexts, our research is focused on the process itself, rather than on optimality. The aim of this work is to verify whether developers have, when they develop software, a behavior comparable to that of stigmergic insects.

Just to introduce with some more detail stigmergy in social insects, we introduce now a model for the behavior of ants when looking for food. In the next section, a similar model will be proposed for libre software development. Figure 1[3] show the paths followed by ants in their search for food. Ants are depicted as small points at the end of their path. The food and the nest are signaled in both parts of the figure. The picture on the left shows the way ants proceeded before finding the food. Once an ant finds food, it uses pheromones to "mark" the way to the corresponding location.

[1]From the Greek 'στιγμα' (mark/sign) and 'εργος' (work)

[2]In an autocatalytic reaction a product of the reaction also acts as a reactant, modifying the speed of the reaction

[3]This figure has been taken from the book 'Swarm Intelligence: From Natural to Artificial Systems" by Bonabeau et al. [9]

Since the moment when food is found by the first ant to the moment shown in the picture on the right, the pheromone path gets stronger and stronger because of the transit of other ants (auto-catalysis), while it gets optimized by a partially random behavior (which is explained afterwards in detail). The randomness explains why even when there is a (possibly optimal) way to the food, there are still some ants which follow an alternate path. This can be observed by the existence of the other gray paths on the right picture that are different from the main path.

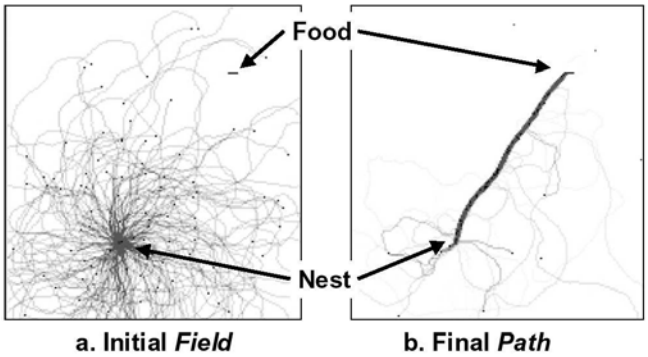


Figure 1. Optimal path from the nest to the food. Ants solve this problem by using stigmergy.

The logic of every entity (in this case an ant) involved in the process is as follows. At the beginning, any ant looks in the environment for information about food, i.e. smelling pheromones that have been deposited by other ants. An ant can follow a certain pheromone mark, depending on whether a random function is below (or above) a given threshold. The threshold depends of course on the intensity of the stimulus: the more intense, the higher (or lower) the threshold value will be, and the more likely the ant will follow it.

Once the ant moves, it can find food or not. If food is found, the ant takes some of it and goes back to the nest, secreting pheromones on its way. If no food is found, the ant just moves on and gets new information from the environment. In other words, the process restarts from the beginning. The whole process can be represented as a flow chart (see figure 2).

This model assumes that the whole system is composed of a set of ants, but that any individual is treated as a single entity. It also assumes that the system is changing dynamically: new stimuli appears in the environment while ants find food and take it to their nest. Pheromones have also an evaporation coefficient, which causes old information to become less and less important with time.

III. Modeling libre software development

Before presenting the stigmergic model for libre software development, we introduce the basic assumption that will lead to it. We consider that developers modify the environment by producing "development activity" (i.e. source code). The production of code is the first step towards building software systems, whose presence may be seen as a stimuli for other developers to work on them. The larger a project (in source lines of code) is, the more likely it will be that a random developer collaborates on it. This

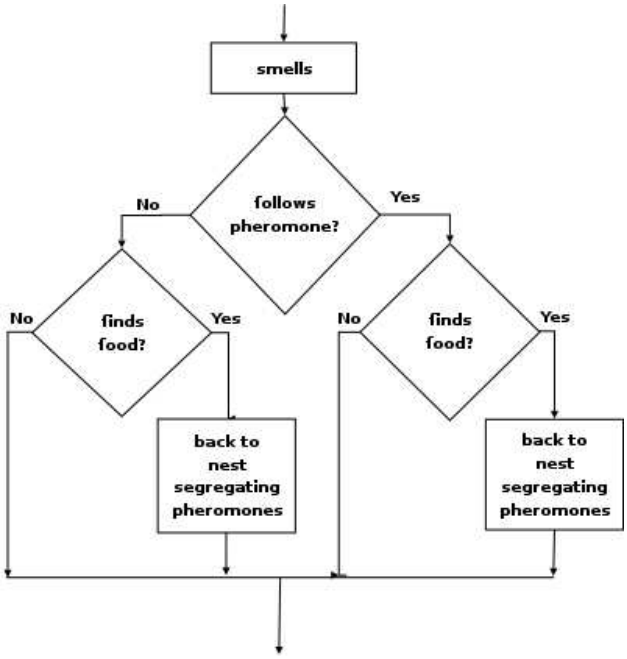


Figure 2. Stigmergic algorithm used by ants to find the optimal way to food from the nest.

is so because large projects get more attention, and may target the needs of more developers (among other reasons).

With this assumption in mind, our hypothesis is that an autocatalytic process, similar to those found in the behavior of colonies of social insects, will be found. Source code and activity around source code are in our model the *pheromones* while libre software projects are the *environment*.

The model for developers should be, in principle, more complex than the one for ants, since the latter make use of 'genetic' knowledge which is quite different in nature to the one used by developers need to create software. For instance, we have to consider program comprehension aspects, such as the learning process and the acquisition of experience. Both will be different when a developer collaborates on a project or not.

Our model is built up on three classes: map (Map), projects (Project) and developers (Ant). The high-level details of these classes are:

- The map is conceptualized as a two dimensional matrix of a given size, n, where each cell corresponds to a kind of software. Cells contain projects and developers. The length of the map, n, is much smaller than the number of developers, N. Any cell may contain as many projects as have been created by developers in it. In summary, this matrix is an abstraction of a software map
- Projects are identified by their name, and their primary characteristic is their size (in number of lines of code). They contain a list of developers which may be active or not on the project. Projects are located in the corresponding cell, according to their application domain and hence cannot change their position. Our model assumes that the larger a project is, the more *stimulating* it will be for developers to work on it.
- Developers are the active agents in the model. They can be identified by their name, and their primary characteristics are their current location in the software map, and their technical skills and experience. Skills and experience will depend on the application domain, and may change in time by using or developing software for that application domain. Developers can move from cell to cell (moving to new application domains) in the software map, but they can only participate in a project in the cell where they are located.

The dynamic part of the model is provided by developers, as described in figure 3. The flow diagram contains several paths which depend on decisions taken by the developer. Those decisions depend on several factors that will be exposed in the following subsections. The high-level characteristics of the model will be presented in more detail in Appendix I, while the implementation details of these decisions will be discussed in Appendix II.

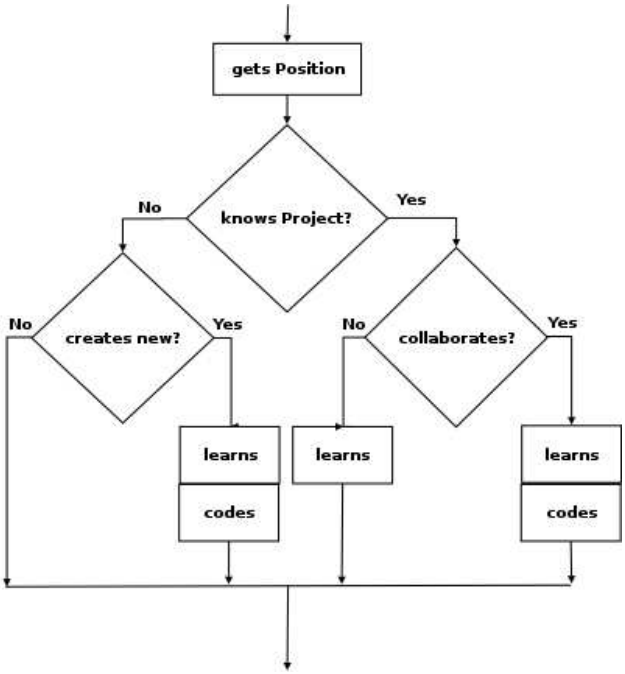


Figure 3. Stigmergic algorithm used to model libre software developers in each turn.

The working of the proposed algorithm is as follows. Developers work in turns. The first action in every turn is to determine the position in the map where the developer will be located. Once that position is known for the current turn, it will be checked whether the developer finds (gets to know) a project in it.

If the developer finds an already existing project, he may collaborate on it, learning (gaining experience) and improving his skills. The amount of work performed will depend both on previous experience and skills, as well as on the time devoted to it. But if the developer, even finding a project, decides not to work on it, he may still learn from it by using its software. In any case, learning by using happens at a slower rate than by collaborating in code production. On the other hand, if the developer does

not find a project, he may decide to create a new one.

All these behaviors are modeled as probabilistic functions, in which a random value has to be above a threshold that may depend on parameters that have to be calibrated carefully. Thresholds are normalized to one and may have maximum/minimum values to avoid that a path becomes impossible to follow. For further details, see Appendix II.

IV. Validating and verifying the model

The parameters and thresholds that have been used in the model have to be configured properly in order to obtain a realistic model. Data from previous research studies, and surveys performed on libre software developers have been used for this purpose. So, for instance, the mean time spent on a project by libre software developers has been taken from several surveys, and productivity has been measured in SLOC, in order to be comparable with the application of the COCOMO cost estimation model [10].

Verification of the model has been performed against results obtained by other research studies. We can classify these results in two sets: facts about developer contributions, and evidences about the software produced. In both cases, with the former calibration, our model shows similar trends to those obtained in these studies.

Regarding developer contributions we have observed whether the model shows a mean involvement, as reported by Hertel [11]; inequality patterns for developer contribution to projects, as found by Koch et al. [12]; whether projects are led throughout the lifetime of the project by several generations of core groups [13]; and whether the number of developers in projects follows a power-law distribution [14]. Other contribution patterns as stated by Mockus et al. on Apache and Mozilla [15], have also been compared with the results we have obtained.

Regarding the size of the developed software in the model, we check whether the average size of software projects remains constant in time, and if we obtain a power-law for project size as some studies on GNU/Linux distributions have thrown [16]. Finally, we look at the software evolution patterns that our model provides and compare it to the current state of the art [17], [18] in this respect.

Verification of the model has been performed against results obtained by other research studies. We can classify these results in two sets: facts about developer contributions, and evidences about the software produced.

Our implementation assumes that the matrix is quadratic and that its size does not change in time. In order to see how the size of the matrix affects the results, we have run our simulation for several values that range from 100 cells (10x10) to 2500 cells (50x50).

The number of developers is another dimension which we have to set, specially as the relation matrix size/number of developers will give an idea of the concentration of developers and probably affect the results. Thus, we simulated with values for number of developers that go from 50 to 10,000. The number of turns (weeks) was set to 750, around 15 years.

In the following subsections, we will compare the results thrown by our simulation and compare it to evidences from previous research studies.

A. Mean time per week

Hertel et al. [11] report from a survey made to 141 Linux kernel developers that the mean time these developers devote to developing libre software is in the mean 12.37 hours per week.

For all the values for the matrix size and the number of developers with which we want to validate our model, the mean time devoted to development laid between 11.4 hours/week and 11.9 hours/week. The statistical mean for probability function of time is 11.71 hours/week.

B. Number of developers and age of a project

Krishnamurthy [22] made an analysis of around 100 mature libre software projects. He stated that the number of developers working on a project was correlated to the age of the project. An analysis of our simulation shows that this is a common trend.

C. Distribution of work

Mockus et. al [15] reported that certain libre software projects as Apache and Mozilla rely on a small group of developers (called the 'core') who control the code base. This group is responsible for around 80% of the contributions. Other research studies have shown this trend for other projects, as for instance Koch for GNOME [12]. From a general point of view, Ghosh et al. found that this happens to be in general for all libre software [23], [24].

Devs	Map Size	Mean	2nd Quintile	4th Quintile
50	100	0.74	0.71	0.91
50	2500	0.53	0.44	0.82
75	100	0.73	0.71	0.91
75	2500	0.50	0.40	0.80
100	100	0.75	0.72	0.91
100	625	0.62	0.56	0.86
100	2500	0.5	0.41	0.81
500	100	0.70	0.69	0.91
5000	100	0.69	0.64	0.88

Table I
GINI COEFFICIENT FOR THE PROJECTS

A way of measuring the inequality in contributions is by using a classical index used in economics for wealth distribution: the Gini coefficient [25] which is tightly related to the Pareto distribution. Table I gives the results from our simulation for the various parameters we have used ("Devs" being the number of developers and "Map Size" the number of cells). The mean of Gini coefficients goes from 0.5 for a setting where there is a low developer concentration to values around 0.75 similar to the ones reported for SourceForge projects[4]. The table shows also values for the second and fourth quintile to give insight about the distribution of data. It can be observed that the distribution is always skewed to values nearer to 1.

D. Power-law of contributors

The distribution of developers among projects has been reported to follow a power-law in a study performed on more than 50,000 projects [14].

[4]See http://libresoft.urjc.es:9080/libresoft/25

Figure 4 is the result of plotting the results from our model for several runs (the ones done with 100 cells). On the vertical axis we can see the number of developers (in log scale) while the horizontal axis (also in log scale) gives the projects.

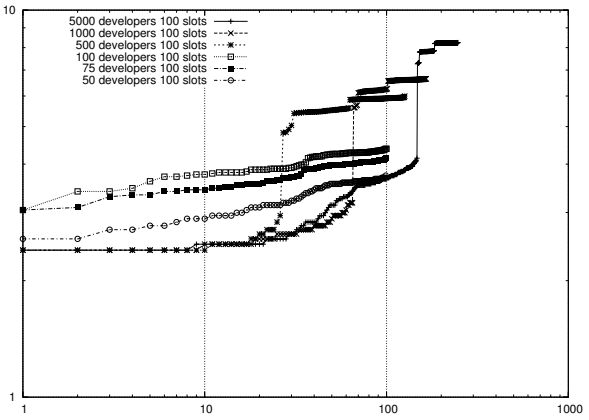


Figure 4. Number of developers in log-log scale.

The figure shows that for all curves we have got two different zones, both with the shape of a power-law (linear in log-log scale). Interesting is the existence of a breaking point between them, meaning that there exists a vacuum of projects with an intermediate number of developers in our model. Once projects achieve certain size, they accelerate themselves enormously. This, of course, does not comply to any study performed on libre software, at least not in this way throwing out such a gap. It seems that this effect in the simulation depends on the concentration of developers, so probably letting the size of the software map grow in time may make this curve grow smoothly as a unique power-law.

E. Power-law of project sizes

Gonzalez-Barahona et al. demonstrated from a study on a GNU/Linux distribution that the sizes of the projects also follow a power-law curve [26].

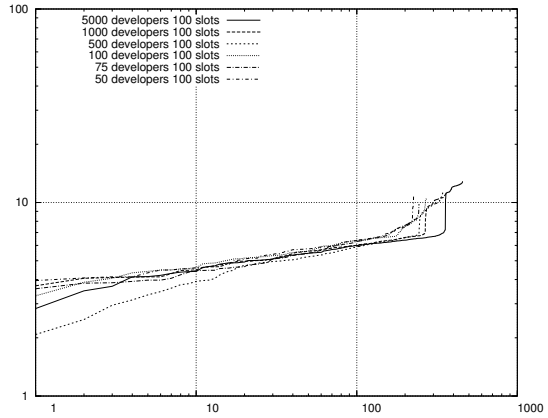


Figure 5. Size of projects in log-log scale.

Figure 5 plots projects in the horizontal (logarithmic) axis and project sizes in the vertical (logarithmic) axis. The resulting curve is clearly power-law for all the cases plotted with a distorting side-effect at the end of the curve possibly due to the same issues as discussed in the previous subsection.

F. Mean size remains constant

Another ecology study performed on GNU/Linux distributions in time threw as results that the mean size of the packages remained constant in time [16].

We haven't observed in the simulations that the mean size of the projects remain constant in time for any of the parameters considered. In our model, what remains constant for all the cases considered is surprisingly the median. But the mean size has a tendency to grow even if new projects are created by developers. This could imply that the creation factor is too high and too few projects are created in our simulation or that having an existing project in a cell makes it very difficult to have other projects there too.

G. Initial size

We also wanted to check if the assumption that the first contribution should be statistically bigger in terms of SLOC is meaningful or not. Therefore we looked at the sizes of packages for the Debian GNU/Linux distribution in time [16]. In Debian 2.0 168 out of 1096 (15%), in Debian 2.1 217 out of 1551 (14%), in Debian 2.2 446 out of 2611 (17%) and in Debian 3.0 739 out of 4579 (16%) are under our mean initial value (400 SLOC), which makes our assumption hard to stay.

But looking at the model, the first quintile gives for different runs with different parameters 295 SLOC, 290 SLOC, 282 SLOC, 283 SLOC. So this result throws out that our model is correct, or even that the 400 SLOC that we have considered as initial mean value is too few.

H. Generations

A study performed on large libre software found that the leading core groups identified by Mockus et al. [15] where not stationary in time, but that generations of core groups in time could be identified [13]. In other words, the ones that started the project do not have to be part of it all the time and newcomers take it over and make the project evolve.

Devs	Map	# Proj	% Proj	% Founder	% 1st Third
100	100	118	39.6%	4.95%	54.43%
100	625	559	36.68%	27.58%	49.1%
100	2500	97	2.69%	44.83%	54.9%
50	100	109	37.46%	9.6%	49.22%
50	625	255	20.88%	40.02%	54.44%
50	2500	5	0.22%	49.92%	54.81%
75	100	111	38.01%	5.95%	54.68
75	625	492	34.72%	33.96%	51.78%
75	2500	23	0.77%	57.76%	59.52%

Table II

LOOKING FOR GENERATIONS IN THE CORE GROUP. CONTRIBUTION OF THE FOUNDER AND OLDEST THIRD OF DEVELOPERS TO THE PROJECT.

Table II gives the result of looking if we can find such a behavior in our model. We have therefore taken projects that have a number of developers bigger than 6 (see column % Proj to see how many projects are affected by this choice) and have seen the percentage of the code that has been contributed by the project founder (% Founder) and by

oldest contributors (% 1st third). The mean values that are shown in the table reflect that the proportion of work made by the founder and the oldest third is in fact important, but by far not predominant. It should be reminded that a stable core group from the beginnings would give values above 80% as discussed in previous subsections.

I. Software growth

There have been a few studies that have investigated the system growth in libre software applications, being specially significant the one performed by Godfrey and Tu on the Linux kernel which showed that it threw a super-liner growth [17] apparently breaking one of Lehman's Laws on Software Evolution [18].

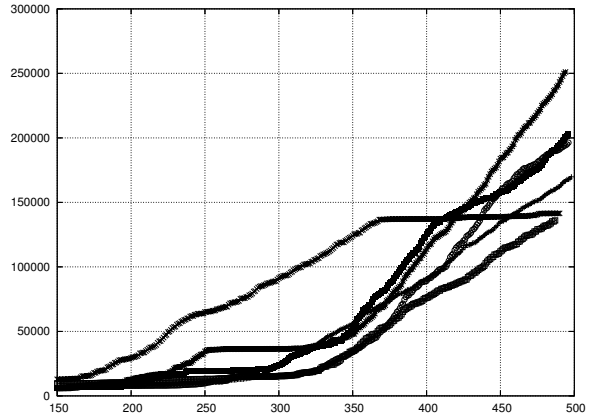


Figure 6. Growth in number of SLOC for some projects.

Figure 6 shows that for the largest projects from one of our analysis, no such super-liner growth can be identified. At most, projects show a linear rate with some trends of super-linearity for some time. Of course, in this case the number of case studies in the libre software world allows us not to make conclusions about if super-linearity is a common trend or only achievable by Linux because of its special circumstances as Godfrey points out [17].

V. Discussion

After presenting the model, we will discuss in this section some of its possible uses, its limits, and some other general considerations about its implications in the software engineering field.

A. Development attracts developers

The effect that is in the core of our model could be summarized as "development attracts developers". This is by no means a strange factor of social interactions: activity is usually an attractor for more activity. This helps to explain from the growth of cities to the success (or not) of cafeterias in a given area. In our case, the attraction by successful libre software projects is clear. Many developers prefer to collaborate in a mature, large and well known project than in a small, hardly known, obscure one. This is reasonable from many points of view. It is more likely that the developer get knowledge about a well-known project, his expectations about becoming known himself are higher, it is more likely that he has some kind of relationship with a current developer (since there are more of them), getting help from peers is more likely, and expectations about success (joining a project which is already a success) are higher.

Probably this is also the reason for the high rate of abandoned or one-developer projects. If a project is small, it is going to have a hard time attracting new developers, which means that no new resources are available for development, that growth is slow or that visibility will also be small and that in turn makes the project not more attractive to developers... On the contrary, if a project reaches a certain size, more and more people know about it, and will eventually consider joining it. This is perfectly consistent with the power-law distribution found in the size of libre software projects and the rich-get-richer effect, for instance.

B. Limits

One of the first questions that arises when presenting a new model is under what conditions it fails to be reasonable or plausible. In this regard, we have to highlight that the main drawback of our model is that it considers no direct communication mechanisms between developers, but this kind of communication does exist in real libre software projects, and is in fact quite important. However, it is also important to notice that the model matches quite well real data, as is shown in appendices. Which would mean, from other point of view, that maybe that direct communication channels have little influence on how developers choose projects, and on how the evolution of human resources available to a project shapes its evolution.

Going further in the limits of the model, it is convenient to notice that there are many factors, very important in the low-level understanding of a project, which are not considered. Some examples are those related to the "external relations" of the project: documentation, blogs, support sites, web sites, etc. All of them have a clear impact on the visibility of the project, and therefore on the attraction of new developers. However, they are not considered in the model.

C. Applicability in proprietary domains

Software development goes beyond libre software and as such it would be interesting to study whether this model is applicable to non-libre (proprietary) software development environments.

Since our model emphasizes the availability of the source code, and in fact considers it as an attraction factor, the case of proprietary software is basically not considered at all at least for traditional in-house developments in companies. In this sense, the model may be helpful to explain how "development communities" work for this kind of companies, although they are usually built upon partnership contracts among companies, many similarities could be observed.

But some parts of the model could be used to understand and even to estimate whether it could be convenient to distribute as libre software some system when a given company wants to quickly promote a large development community around it. Besides the explicit consideration that the software company should try to attract developers to enter an autocatalytic reaction, other questions could be simulated with this model. In this regard, many

companies that want to invest in a software developed by the community do not know if the effort they should devote to the project is by hiring developers from this community or to contract new developers that should integrate themselves into the project. The former provide probably over a system-wide knowledge and have already experience and expertise on the project, while the latter add new developers to the project (which makes the project more *attractive* in our model) although they require over a software comprehension phase to gain understanding over the system and become productive.

In recent years, some large software companies have started to look at the libre software phenomenon and to adapt their strategies to it even if not releasing their software under a libre software license. Their objective is to leverage the libre software development process as described in this paper (autocatalytic process, to benefit from external contributions, etc.) without losing the control of the project that copyright law grants them[5]. It is difficult to know the impact of the license used by a software project on the willingness of new developers to join in. In any case, licenses have different implications which can be seen as positive or negative for different developers, which in our model could increase or decrease their particular thresholds to join a project.

D. Social and cultural factors

Another limitation that our model has is that we consider developers being too homogeneous. In this sense we only take into account skills for characterizing developers and hope that the statistical considerations introduced in the model will make heterogeneity arise among developers and the model more realistic. However, some other factors such as geography or cultural affinities could also be important for the selection of a project to join in and of course the barrier of entry for a project will be different depending on these factors. For instance, it seems reasonable that if there is a large group of developers of a certain project in a geographical area, new developers in that area are more likely to join that one. Or that language could impose barriers for entering a give project. However, we do not know about studies which show whether this actually happen. Our model is global in that sense: any developer with a certain set of skills is equally likely to join a given project, with independence of where he lives, which languages he speaks, or any other peculiarity he may have.

Considering after the discussion in the previous paragraph, that the pool of developers available in a certain domain is global, the model will only work when there is a certain number of them. If there are too few, no stigmergic process is possible. We have not studied that lower bound, but it is probably above the usual number of developers in a given domain that most companies have hired. This would be another reason why these stigmergic processes does not happen in proprietary in-house software environments.

E. Stigmergy, the bazaar and the cathedral

The classical model for libre software compares it to a bazaar as there is a lack of formal rules, hierarchies and mechanisms for "imposing" what an individual developer will do. In opposition, we find the traditional way software is developed where processes, roles and tasks are specified and which has been compared to the way cathedrals were built centuries ago [2].

If we consider our stigmergy model in the context of the bazaar, it shows some common properties. The model behaves in a certain way as a bazaar, with no formal rules, no formal hierarchy, and no task-imposition mechanisms (in our case, which project a developer joins). However, our model is more appropriate than the bazaar model when comparing it to the cathedral. This is because although the bazaar model explains concisely the exchange of code and knowledge in a non-controlled environment, its analogy lacks of a very important element: a final (tangible) artifact produced by all those interactions as we have in the case of software (its source code). This is not the case for the cathedral model as at the end of the process we have, obviously, a cathedral.

And that is precisely the reason why we think our model is a better analogy for the development in libre software environments as stigmergy may also lead social insects to build complex structures. This is the case for termite nests which could in fact be considered literally as cathedrals made of clay. This means that with the stigmergic model we have a complete analogy that allows to compare the construction of cathedrals as it was done by humans several centuries ago (and which follows process similar to traditional in-house software development) with the construction of termite nests (which obeys to behaviors that can be found in the development of libre software systems). In addition, our model illustrates that developers need to be stimulated in order to make such a development process happen while the bazaar analogy does not consider how the bazaar comes to be.

In other words, our model explains how even if the final product is the same (cathedrals, ie. a software product), the process that has made it possible is completely different. It is actually not about what is produced but how it is produced.

F. Information hiding

Finally, we cannot help but enter into a historical debate about how information should be managed in software development processes: the classical Brooks [19] vs. Parnas [20] argument about information hiding. The latter proposed in the early seventies that there should be some design and implementation decisions that software developers should hide in one place from the rest of the program which was criticized by the former.

Our model is certainly more oriented towards Brooks' position. Information (source code) should not be hidden as it is the stimulus that makes an autocatalytic reaction possible and the whole software development phenomenon successful.

Interestingly enough, the 20th anniversary edition of the "Mythical Man-Month" Brooks states that this was one of the seldom errors in the original version. Probably from the pure software development methodology point of view, information hiding is more appropriate and is a practice that should be followed. But, as we can see from the model

[5]This is the case, for instance, of Microsoft's Shared Source Initiative or the Java Community created around Java by SUN.

presented in this paper, software engineering sometimes has to consider other factors as development does not happen in a vacuum. Software projects that intend to be based on stigmergy should manage information hiding in a balanced way to assure software quality and developer attraction at the same time.

VI. Conclusions

This paper is a first step towards understanding the social and computer-mediated interactions that yield, as a final product, a libre software project. We have therefore proposed and verified an analogy with how social insects perform large-scale works by means of indirect communication entities that act as stimulants.

Our primary conclusion is that libre software development can indeed be modeled as a stigmergic one, at least with respect to how developer effort is allocated to projects, and to how this affects in the evolution of projects themselves. The model we have proposed shows patterns similar to those reported in real world studies on libre software, although finer calibration and further discussion on the variables and threshold values is necessary. This means, for instance, that the individual productivity may not be as important as the total production as a community; or that stigmergic mechanisms could be used in order to increase productivity at the project level.

On the other hand, our model raises a set of interesting questions the we have briefly discussed. We have seen that development attracts developers and that this activates a rich-get-richer effect which makes successful projects even more successful and attractive to new developers. We then have discussed the limitations that our model exhibits and have presented our doubts about if this a stigmergic process could be applicable in proprietary environments because of the lack of critical mass and the fact that developers may have different threshold values for more restrictive licenses. Finally, we have indicated how stigmergy gives a clearer analogy of some of the factors that characterize libre software development than analogies that are commonly used and we have discussed how our model fits into the classical information hiding debate.

Future research could focus on studying how the modification of some of the parameters may affect the development of software in the libre software world. This could be the case of a company wanting to hire developers for a libre software project. An interesting extension could be to study how other 'communities' present in libre software development (i.e. translators, documenters, etc.) can be included in a model like the one presented.

References

[1] W. Scacchi, "Free and open source development practices in the game community." *IEEE Software*, vol. 21, no. 1, pp. 59–66, 2004.

[2] E. S. Raymond, "The cathedral and the bazar," *First Monday*, vol. 3, no. 3, 2000.

[3] R. Ferenc, I. Siket, and T. Gyimóthy, "Extracting facts from open source software." in *Proceedings of the International Conference in Software Maintenance*, Chicago, IL, USA, 2004, pp. 60–69.

[4] I. Samoladas, I. Stamelos, L. Angelis, and A. Oikonomou, "Open source software development should strive for even greater code maintainability," *Communications of the ACM*, vol. 47, no. 10, October 244.

[5] J. W. Paulson, G. Succi, and A. Eberlein, "An empirical study of open-source and closed-source software products," *Transactions on Software Engineering*, vol. 30, no. 4, April 2004.

[6] D. German, "An empirical study of fine-grained software modifications," in *Proceedings of the International Conference in Software Maintenance*, Chicago, IL, USA, 2004.

[7] P.-P. Grassé, "La reconstruction du nid et les coordinations interindividuelles chez bellicositermes natalensis et cubitermes sp. la théorie de la stigmergie: Essai d'interpretation du comportement des termites constructeurs." *Insectes Sociaux*, no. 6, pp. 41–81, 1959.

[8] T. Susi and T. Ziemke, "Social cognition, artefacts and stigmergy: A comparative analysis of theoretical frameworks for the understanding of artefact-mediated collaborative activity," *Journal of Cognitive Systems Research*, no. 2, pp. 273–290, 2001.

[9] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Articial Systems*. Oxford University Press, Inc., 1999.

[10] B. Boehm, *Software Engineering Economics*. Prentice Hall, 1981.

[11] G. Hertel, S. Niedner, and S. Hermann, "Motivation of software developers in open source projects: An internet-based survey of contributors to the Linux kernel," *Research Policy*, vol. 32, no. 7, pp. 1159–1177, 2003.

[12] S. Koch and G. Schneider, "Effort, cooperation and coordination in an open source software project: GNOME," *Information Systems Journal*, vol. 12, no. 1, pp. 27–42, 2002.

[13] J. M. Gonzalez-Barahona and G. Robles, "Unmounting the "code gods" assumption," in *Proceedings of the Fourth International Conference on eXtreme Programming and Agile Processes in Software Engineering*, 2003. [Online]. Available: http://libresoft.dat.escet.urjc.es/html/downloads/xp2003-barahona-roble%s.pdf

[14] K. Healy and A. Schussman, "The ecology of open-source software development," University of Arizona, USA, Tech. Rep., Jan. 2003.

[15] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two case studies of Open Source software development: Apache and Mozilla," *ACM Transactions on Software Engineering and Methodology*, vol. 11, no. 3, pp. 309–346, 2002.

[16] J. M. Gonzalez-Barahona, G. Robles, M. Ortuño-Perez, L. Rodero-Merino, J. Centeno-Gonzalez, V. Matellan-Olivera, E. Castro-Barbero, and P. de-las Heras-Quirós, *Free/Open Source Software Development*. Idea Group Inc., 2004, ch. Analyzing the anatomy of GNU/Linux distributions: methodology and case studies (Red Hat and Debian).

[17] M. W. Godfrey and Q. Tu, "Evolution in Open Source software: A case study," in *Proceedings of the International Conference on Software Maintenance (ICSM 2000)*, San Jose, California, 2000, pp. 131–142.

[18] M. Lehman, J. Ramil, P. Wernick, and D. Perry, "Metrics and laws of software evolution - the nineties view," in *Proceedings of the Fourth International Software Metrics Symposium*, Albuquerque, NM, USA, 1997.

[19] F. P. Brooks, *The Mythical Man-Month*. New York: Addison Wesley Longman Inc., 2001.

[20] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Communications of the ACM*, vol. 15, pp. 1053–1058, 1972.

[21] R. A. Ghosh, R. Glott, B. Krieger, and G. Robles, "Survey of developers (free/libre and open source software: Survey and study)," International Institute of Infonomics. University of Maastricht, The Netherlands, Tech. Rep., June 2002.

[22] S. Krishnamurthy, "Cave or community? an empirical examination of 100 mature open source projects." *First Monday*, vol. 7, no. 6, 2002.

[23] R. A. Ghosh, G. Robles, and R. Glott, "Software source code survey (Free/Libre and Open Source Software: Survey and Study)," International Institute of Infonomics. University of Maastricht, The Netherlands, Tech. Rep., June 2002.

[24] R. A. Ghosh and V. V. Prakash, "The orbiten free software survey," *FirstMonday*, vol. 5, no. 7, May 2000.

[25] C. Gini, *On the Measure of Concentration with Espacial Reference to Income and Wealth*. Cowles Commission, 1936.

[26] J. M. Gonzalez-Barahona, M. A. Ortuño Perez, P. de las Heras Quiros, J. Centeno Gonzalez, and V. Matellan Olivera, "Counting potatoes: The size of Debian 2.2," *Upgrade Magazine*, vol. II, no. 6, pp. 60–66, Dec. 2001.

APPENDIX

I. The conceptual model

In this appendix we will introduce the high-level characteristics of our model.

A. Getting a (new) position in the map

The developer gets at first a random position in the software map. This position may change at the beginning of every turn. This is done following a probabilistic function that depends on several parameters, which are explained below.

There is a natural dullness to remain on the same position, given by a minimum value for the threshold. There should be a maximum possible value for the threshold, so there should always be a chance of moving to another places in the software map.

$$threshold = f(dullness, skills, experience)$$

Hence a developer moves to a new (random) position if:

$$random\_value(0,1) > threshold$$

B. Finding a project

Once a developer is on a given position, he starts looking for projects. Some parameters will make finding a project more probable: being a large project, a huge number of developers participating and the time the project exists. All of them make the project more easy to find, up to a minimum threshold value that assures no project is known by anyone anytime. We will hence assume that the threshold depends on an initial value which will give the probability of finding a project even being small, with few developers and new and that will decrease as these parameters change until a minimum value is achieved:

$$threshold = f(size, developers, age)$$

Hence a developer knows (finds) a project if:

$$random\_value(0,1) > threshold$$

C. Participating in a project

Skills and experience a developer has on a position are going to be the parameters that affect the calculation of the threshold value. Again we will have a maximum value which will make it possible that even without skills and experience a developer works on a project, as well as a minimum value which implies that working on a project may not happen even having high values of skills and experience. A minimum value could be selected proportional to vacation time or similar.

$$threshold = f(skills, experience)$$

Thus a developer collaborates on a project if:

$$random\_value(0,1) > threshold$$

D. Creating a project

In this case we assume that the developer has a natural inclination to create a new project, which depending on the implementation will be high or low. Natural inclination will be affected by skills and experience on other places in the software map, specially if they are located near the current one.

$$threshold = f(inclination, skills, experience)$$

In our model a developer creates new project if:

$$random\_value(0,1) > threshold$$

E. Amount of work (code) performed

By coding a developer produces lines of code for a project. The amount of work that is done depends on the time that is committed to do it as well as on the skills and experience of the developer. A developer acquires experience by working on a project. Experience is measured in terms of source lines of code and depends on the application domain (i.e. the position in the software map).

$$SLOC = f(skills, experience, time)$$

F. Learning rate

Learning can be done by submitting code to a project, or just by "using" a project. "Using" should be understood in the advanced user way, so that possibly reading the mail archives, submitting bugs and reading even technical documentation is comprised. By learning a developer acquires skills.

$$skills = f(usingordeveloping)$$

The function for skills as well as the one for experience could not be simply additive, but have also into account a decreasing of both parameters as time passes.

APPENDIX

II. Implementation of the model

The stigmergy model presented in this paper has been implemented with a Python-based program called pyStigmergy[6]. The main implementation details are explained and discussed in this appendix.

A. Getting a (new) position in the map

The natural dullness to remain on the same position (used as a minimum value) has been set to 0.7, while the maximum has been given a value of 0.99. Choosing 0.99 means that if turns are weekly, a developer would at least change his position in the mean every two years.

$$random\_value(0,1) > 0.7 + \frac{skills}{100} + \frac{experience}{10000}$$

The normalization values have been put to 100 for skills and 10000 for experience. More insight about these values (and hence their normalization) is given in subsection F.

B. Finding a project

The maximum value of the threshold for finding a project has been set to 0.8, which in fact means that finding a project is by default difficult. The minimum value has been set to 0.01.

Size, number of developers and age have been normalized into formulas to give number that range from 0 to circa 6. All formulas are monotonic increasing and have the property that they new code/developers when the project is small affects the visibility more than when the project is already large.

$$size = \frac{ln(project\_size)}{2}$$

$$numDev = \frac{\sqrt{project\_numDev}}{2}$$

$$age = project\_age * 0.05$$

Age has been taken to be linear and measured in weeks with a factor of 0.05. Age is given by the number of turns a project has been worked on by at least one developer, being also a proxy for a release in the 'release often' libre software fashion [2].

As we can see from the next equation, the three parameters are treated equally by summing them up and by dividing them by a value that is dependent on the constant. The 64 has been taken as a normalization value and is the result of 4^3, which is a sort of mean of all parameters.

$$random\_value(0,1) > 0.8 - \frac{size + numDev + age}{(1-0.8) * 64}$$

[6]The program is libre software, licensed under the GNU General Public License (GNU GPL), and can be downloaded from `http://libresoft.urjc.es`.

C. Participating in a project

We suppose that the maximum value is given by a constant that has been set to 0.5, meaning that a newcomer has the same probability of working on the project than not doing it. Skills and experience (properly weighted in order to become normalized) will lower the threshold up to a minimum value of 0.01.

$$random\_value(0,1) > 0.5 - \frac{skills}{1000} + \frac{experience}{50000}$$

D. Creating a project

We assume that developers are not very inclined to create a new project, so we have put a constant threshold value of 0.8 that should be surpassed. Although the conceptual model discusses other parameters, we have not considered them for our implementation.

$$random\_value(0,1) > 0.8$$

E. Amount of work (code) performed

The amount of work performed depends on the time devoted to a project. Based on the FLOSS survey [21] which was answered by over 2000 respondents, we have chosen a probabilistic function that descends linearly from 1 hour to 40 hours. The maximum involvement is 40 hours/week. Involvement is calculated randomly from 0 to 1 and then transformed to the previous probability function by means of the following equation:

$$time = 40 - \sqrt{1600 - 1600 * involvement}$$

The production of the developer depends on its skills and experience in this position. The maximum value for skills * experience is 50.

Finally the worked done (measured in SLOC) is given by the multiplication of all the parameters. As the multiplication of skills and experience as well as time have both an upper bound, the maximum amount of SLOC that can be produced by a developer in a single week has also an upper limit: 2000 SLOC[7].

$$worked = round(skills * experience * time + 0.5)$$

Creating a project works differently. We have supposed a random amount of code is added at first because of the programming structure and an initial size that the project has to have before being released:

$$worked = rand(0,1) * 800$$

This means that we assume that contributing a new project has in mean 400 SLOC more than a contribution to an existing project.

F. Learning rate

If a developer works (submits code) to or uses a project, his skills get increased following this equation:

$$skills = skills + factor * rand(0,1)$$

where the value of the factor is 2 for developing and 0.5 for "using" the software.

[7]Although this is the maximum value and it may be obtained only under certain, very specific circumstances it appears to be very big (compared for instance to the estimation given by the COCOMO model [10]). In section **??** we discuss these parameters.

Understanding Team Forming in Software Development

Silvia T. Acuña
Escuela Politécnica Superior
Universidad Autónoma de Madrid
silvia.acunna@uam.es

Marta Gómez
Escuela Politécnica Superior
Universidad San Pablo-CEU
mgomez.eps@ceu.es

Natalia Juristo
Facultad de Informática
Universidad Politécnica de Madrid
natalia@fi.upm.es

Abstract
People are a fundamental and critical concern in software development success or failure. There is research that takes this aspect into account and incorporates people into the software process. This is done by analysing people individually and establishing relationships with the activities that are performed within the project. Then again there is agreement on the fact that these people work together to perform interdependent development tasks, and these group interrelationships are complex. This leads to the need to examine software developer team forming.
This article presents on-going research into the understanding of the complex structure of software development team forming. First, we present the components of this structure. Second, we characterise the current two development strategies, agile and heavy-weight, based on given team behaviour components. Finally, we set out a cross-experiment to empirically examine the relationships between these team behaviour components and the different types of software process. The importance of studying team forming in software development is that if we can find out what sociological and psychological factors improve development team performance, managers will be able to use this knowledge to form better teams for each way of developing software.
Social psychology is now researching team forming considering a series of factors that have been found to affect team performance. On the one hand, this work analyses team member personality factors and knowledge, skills and abilities (KSA) and, on the other, these analyses are interrelated with task characteristics. This is because team performance cannot be reliably predicted from team personality composition alone, but rather depends on the interactive effects of both team personality composition and team task characteristics. Moreover, this relationship involves other factors, like interactions between people, including conflict, cohesion and climate.
To be able to apply the results of this research to software development, however, data need to be collected that indicate how people/task/team behaviour relationships influence software development team performance and satisfaction to find heuristic rules on team forming.
Keywords: Software process modelling; Agile and heavy-weight software processes; Team forming; Team personality; Task characteristics; Team climate

1. INTRODUCTION

This article is about teams and the collective effort of individuals working together towards a common goal; more specifically, teams that develop software. Here, a team is defined as "a number of individuals brought together for a certain task, goal or objective, engaged in frequent face-to-face interaction to execute a task, while the individuals are (mutually) interdependent on each other with regard to the outcome of the task and its execution" (Katzenbach & Smith, 2001).
We attempt to give an understanding of what factors influence performance differences among development teams, arising, as a function of their composition, in terms of individual personality factors, task characteristics and team behaviour. We have two goals: a) we characterise the two software development strategies now in use: agile process models and heavy-weight process models according to task features and team behaviour aspects and b) we set out a cross-experiment to empirically study what type of people/ people combinations (team) feel more at home with and perform better in different software process types.
Most of the work on team forming in the field of psychology analyses the relationships between personality factors and task characteristics (Molleman et al., 2002; Barry & Stewart, 1997; Hackman & Oldham, 1980). Research has also been done towards understanding the effects of multifunctionality in teams (Molleman & Slomp, 1999). It is accepted that team performance cannot be reliably predicted from team personality composition and task characteristics alone, but rather depends on the interactive effects of both team personality/task characteristics composition and team climate characteristics. Teamwork is affected by what preferences and perception the team members have as to how the tasks should be performed within the team, that is, of what the climate is like. Team performance depends on the personality distribution within a team and the individual team members' ability to handle and work in a particular type of climate.
To define team forming heuristics for the software process, we need to establish relationships between individual factors (personality and KSA), aspects of the tasks and the team climate to improve the performance and satisfaction of the people working together.
Psychologists collect data to ascertain what team forming strategies are applied in a particular domain. These studies are non-intrusive and call for measurements to be taken of climate, personality factors, KSA and task characteristics before, during and after software project development. This will provide data that could be used for exploration, conjecture and to infer correlations for the variables to be considered in software development team forming.
This paper is structured as follows. Section 2 describes related work on individual factors in the software process. Section 3 presents approaches to team forming from the field of social psychology. Section 4 describes the compo-

nents, aspects and values of the team behaviour and effectiveness model. Section 5 describes research into team forming in software development. Section 6 characterises the two development process types: agile and heavy weight, based on task features and team behaviour, and describes the cross-experiment run to fit the best people/team behaviour to each software process type. Conclusions are set out in section 7.

2. INDIVIDUAL FACTORS IN SOFTWARE DEVELOPMENT

People are considered as one of the most important and critical factors within software development with regard to project success or failure (DeMarco & Lister, 1999). In actual fact there is widespread recognition that software process productivity and efficiency is critically dependent on human and social factors (Boehm et al., 2000). A number of studies have been conducted on how to bring people into software development. Most of this research examines the individual qualities of the people involved in the software process, considering personality factors and the competencies required according to the characteristics of the task to be performed, etc.

Acuña and Juristo (2004) have incorporated behavioural competencies and capabilities into the software process, defining the capability/people and capability/role relationships within a development project. These relationships are the basis for determining the capabilities of the development team members to then assign people to role performance, depending on their capabilities and the capabilities required by roles. Turley and Bieman (1995) also discussed the standard skills and personality traits of software engineers and roles in software engineering. Some studies have used standard personality tests, such as the "Sixteen Personality Factor Questionnaire" (Moore, 1991), to define personality profiles of application programmers, application systems analysts, technical programmers and software services managers. Other studies, such as the one by Wynekoop and Walz (2000), try to find the personality traits of software engineers empirically by developing a model of six personality traits applied to software development. Kellner et al. (1999) described how the interrelationships between the human, technical and economic aspects of software projects could be modelled and simulated. Finally, Koontz and O'Donnell (1972) defined five basic principles for software personnel management that provide guidelines for improving the staffing situation of software companies to achieve production goals.

This is research into individual factors of unquestionable importance. Indeed, these factors are part of the process areas of levels 2 and 3 of the People-CMM (Curtis et al., 2001). Nevertheless, while formalised mechanisms for performing these activities are a must, they are not sufficient, as people work together, and this team aspect has a decisive influence on the results of software production, which is basically a social activity.

Citing DeMarco and Lister (1999), "Most software development projects fail because of failures with the team running them". We need to take a step further and examine the development team, its interrelations and characteristics to better ascertain what factors influence team performance in software development.

3. PSYCHOLOGICAL STUDIES ON TEAMWORK

From research in the field of psychology into team behaviour and team forming in particular, we can say that there are three main approaches: KSA approach, structural contingency theory and team climate inventory theory.

3.1. Knowledge, Skills and Abilities (KSA)

Groups usually exist for a reason, and in organisational teams, this reason is to accomplish an organisationally relevant goal. The team can only accomplish this goal if the individual members are qualified enough to contribute to this goal. In other words, the individual members should have the necessary KSAs to be able to work on one or more of the tasks that need completion if the goal is to be achieved. But task-related KSAs alone are not sufficient. Individuals also require interpersonal skills for working with others in a group, and knowledge of, for example, the group's norms such as proper behaviour. The possession of KSAs by individuals in groups is an almost unwritten assumption, since, as an individual is in a group, he or she must have the required KSAs for functioning in a group. It would make little sense to have individuals in a football team who cannot "pass the ball to a team mate". Some studies have been conduct towards gaining an understanding of the effects of multifunctionality in teams, that is, the number of different tasks a worker has mastered (Molleman & Slomp, 1999; Van den Beukel & Molleman, 2002).

3.2. Structural Contingency Theory

Structural contingency has been much researched at organisational level and is now being adapted to team level (Hollenbeck et al., 2002). This theory essentially addresses task characteristics and their match with the team. It is a socio-technical approach, considers both the social and the technical parts and its aim is to match the relevant questions at team level.

Research conducted by Molleman is based precisely on this theory, suggesting the approach of a people-to-team fit by task characterisation (Molleman et al., 2002). This approach proposes characterising the people/task relationship. In other words, the personality traits of the people (conscientiousness, emotional stability, openness to experience, etc.) and the characteristics of the tasks (autonomy, interdependence, etc.) that will lead to a positive correlation in the team results should be determined. This approach is supported by the work of Molleman and establishes a relationship of moderation: the task characteristics moderate the people-team fit.

Other researchers have also pointed out that the task plays a moderator role with regard to of team member characteris-

tics and performance. Hackman and Oldham (1980) discussed the extent to which interpersonal skills contribute to team performance depending on whether group tasks call for more or less interpersonal relations. Another factor is growth needs if the task is not routine and offers a learning opportunity. Molleman and Slomp (1999) suggest that some people may feel uncomfortable if the tasks are ambiguous, whereas others may consider them a challenge.

3.3. Team Climate Inventory Theory

Team climate is supported by Anderson and West's Team Climate Inventory (1994). There are two possible approaches for achieving the people-team fit. The first would establish people's climate preferences individually and the second approach would characterise the team climate through the perceptions of the team members. In both cases, the technique used is the team climate inventory questionnaire (Anderson & West, 1998; Anderson & West, 1999).

The first team climate inventory theory approach matches people to teams according to what climate preferences they have. It is concerned with defining the climate aspects involved in task performance and selecting people who have preferences for the defined climate aspects. This approach aims to select people who would feel more comfortable within the team by investigating what preferences the person in question has by asking questions like "do you like discussing ideas openly in the work team?"

The second team climate inventory approach fits people to the team according to climate characterisation (Burch & Anderson, 2004). A relationship of mediation is established, that is, what is the effect or to what extent is climate mediating the person-team fit. In this case, team climate is characterised by asking questions like "does the project manager give you the chance to discuss and exchange ideas openly, etc.?" Having determined the perceptions on climate, the preferences of each team member can then also be analysed to fit the person to the team.

4. TEAM BEHAVIOUR AND EFFECTIVENESS MODEL

Team forming and team behaviour is composed of four basic components: a) people, b) tasks, c) team behaviour and d) outputs. Figure 1 shows the relationships between these components, as well as the primary aspects of each one. These relationships are very dynamic in terms of variety and change, which makes the group structure open and complex. Let us examine each of these components, as well as their aspects/subaspects and associated values.

4.1. People

Most of the research conducted appraises the individual aspects of the people involved in team forming. Specifically, it analyses the required personality factors and KSAs depending on the characteristics of the task to be performed, as well as preferences for a team climate.

Personality factors determine personal preferences, opinions, attitudes, values and characteristics. In short, everybody has a different personality topology, which is what makes this person a different individual. Studies conducted in the field of psychology have converged on a range of five personality factors. These five personality factors are covered by the Five Factor Personality Inventory, also known as the "Big Five" (Digman, 1990; Barrick & Mount, 1991; Hendriks, 1997), which has come to be the dominant topology in a lot of research, including the field of team forming.

The Big Five questionnaire identifies the five fundamental dimensions of human personality:

- Extraversion, inherent in a confident and enthusiastic view of many, mainly interpersonal, aspects of life.
- Agreeableness, altruistic concern and emotional support for others.
- Conscientiousness, proper to a perseverant, scrupulous and responsible behaviour.
- Emotional Stability, a broad-spectrum trait including characteristics such as the ability to deal with the negative effects of anxiety, depression, irritability or frustration.
- Openness to Experience, especially intellectual openness to new ideas, values, feelings and interests.

Research conducted so far in social psychology point to three primary personality traits of the Big Five as being relevant for effective team operation and results (Molleman et al., 2002): conscientiousness, emotional stability and openness to experience. However, we will use the five personality factors to determine which combinations should be taken into account in agile and heavy-weight software development team forming.

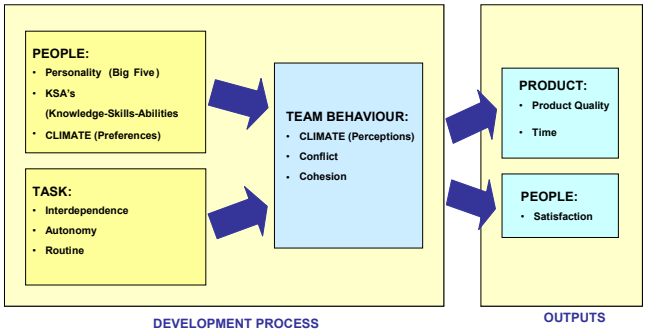


Figure 1. Team Behaviour and Effectiveness Model

Other individual factors that play a role in team forming are individuals' group KSAs:

- Knowledge on how to work in teams and achieve a common goal.
- Skills linked to specific experiences like people's expertise in and strategy for relating within the group and with other groups.

- Abilities, such as the ability to reason, synthesise, analyse, needed to achieve personal and team goals.

For the people component, another aspect that plays a role in team forming, apart from personality and KSA aspects, is what preferences each team member has on the work climate, that is, what environment he or she likes to work in. The Team Climate for Innovation Questionnaire is used for this purpose (Anderson & West, 1998; Anderson & West, 1999). This questionnaire can be used to select people with similar preferences and is, therefore, known by the acronym TSI (Team Selection Inventory) (Anderson & Burch, 2003).
The Team Climate for Innovation Questionnaire is also used to determine the real climate in development teams. This aspect is analysed later under the team behaviour component discussed in section 4.3. Both aspects, preferences for a work climate and perception of the team work climate, measure four subaspects that will also be detailed in section 4.3.

4.2. Task

Task is understood as "a set of specifications identifying the goal that is to be achieved and the procedures that an individual or group may employ when attempting to achieve it" (Steiner, 1972). Task characteristics influence employee performance. There are several task topologies: Steiner (1972), McGrath (1984), etc. For example, Hackman and Oldham's topology (1980) is one of the most widely accepted for individual employee well-being and directed at motivating aspects of the task:

- Skill variety: the different skills necessary to accomplish a job.
- Task identity: the degree to which an individual completes a "whole" product or piece of work rather than just a small part.
- Task significance: the impact that the work has on the lives of others.
- Autonomy: the independence the employee (or group) has in planning and doing the job.
- Feedback from the job itself: the manner in which the job provides feedback about the employee (or group) performance.

Several pieces of research (Breaugh, 1985; Evans & Fischer, 1992; Molleman et al., 2002; Hollenbeck et al., 2002) have revealed that the task characteristics that influence teamwork results are: autonomy, interdependence and routine.

i) Interdependence: task interdependence refers to a situation in which the process and result of one task affects the process and result of other tasks. This interdependence is reciprocal, which means that people (employees) are mutually interdependent. Mutual interdependence means that decisions on "what", "how" and "when" taken by one team member affect the decisions of his or her colleagues and vice versa. Therefore, mutual interdependence calls for communication, cooperation and collective decision making within the team. The possible values for team task interdependence are: Interdependence and Independence.
ii) Autonomy: this refers how much freedom a team has to make decisions on goals ("what"), work methods ("how"), delivery planning ("when") and work distribution among team members ("who"; Molleman, 2000). The possible values for this aspect are: Autonomous and Reliant. Autonomous teams are less controlled and their work is usually less structured. If there are a lot of people making decision autonomously in the team tasks, this is an indication that the team members have a lot of freedom to develop different activities depending on their personality (Barry & Stewart, 1997). Consequently, both team task interdependence and autonomy will moderate how the employee characteristics are related to the team results.
iii) Routine: this refers to the level of non-routineness of the group task. Non-routine tasks are characterised by their uncertainty with regard to operational procedures (how to perform the task), task requirements (what is needed to perform the task), task distribution (who works on what part of the task) and environmental demands (what is the goal/objective of the group), as well as the likelihood of change in these elements, and unique problems that can appear as a result of these aspects of uncertainty (Breaugh, 1985; Evans & Fischer, 1992). The possible values for this aspect are: Routine and Creative.

4.3. Team Behaviour

Several papers (Tuckman, 1965; Steiner, 1972; McGrath, 1984; West, 1990; West & Anderson, 1996; Anderson & West, 1998; Larson & LaFasto, 2001) point to the following team processes as being variables involved in team forming relationships:

i) Cohesiveness: a measure for a group's entitavity (or groupness). It is the result of all forces acting on the members to remain in the group. A team's cohesion is an indicator of its strength. It is an essential characteristic of teams, since team members are more willing to collaborate if the ties that bind them to the team are stronger. The possible values for this aspect are: Cohesive and Cooperative.
ii) Conflict: is a struggle, clash or state of opposition between opposing forces, ideas or interests. This is indicative of the number of interpersonal, intragroup or intergroup behaviours that can be considered a conflict. Wherever a number of individuals work together in a group towards a common goal, some form and level of disagreement or animosity is likely to occur, irrespective of fate. The possible

values for this aspect are: Compulsion and Negotiation.

iii) Cooperation: willingness of an individual to put in effort, either mental or physical, on behalf of the group towards achieving the group's goal, consisting of behaviour that yields maximal joint profit for all the parties involved. Cooperation is the strength of a team derived from pooling the efforts of a number of individuals together to work towards a common goal or task.

iv) Communication: is essential to any team, as a means of distributing information between team members. Communication is the process by which members generate cohesiveness in the form of shared meaning structures.

There are many other team processes, and there is no definite list. As the focus here is on the match between people/team and the different ways of developing software, we have selected two team behaviour aspects, namely, Cohesion and Conflict, which, according to Yang & Tang (2004), are most closely linked to these matches as regards better or worse team performance. Admittedly, the selection might appear to be somewhat arbitrary, and there is a lot to be said for including other team behaviour aspects, but we chose a limited number of team behaviour aspects for practical reasons (length of the questionnaire and efficiency of the model).

Another team behaviour aspect that also has an impact on the team is team climate. As mentioned earlier, team climate refers, on the one hand, to the preferences as regards the work setting and, on the other, to the perceptions on what "life" in the group is like. For example, trust in the team, conflict resolution ability, that is, capability for removing barriers to goal attainment, problem-solving ability, maintenance of a productive interaction between team members and the capability to overcome obstacles standing in the way of team effectiveness are all elements of the climate that mediate between people and the activities they perform in a team and team effectiveness. To be effective team members, therefore, people need an understanding of internal and external team dynamics, as well as of other group types.

The team behaviour component considers the second subaspect of team climate, that is, what perception each team member has of the climate in which he or she is working or has worked. The Team Climate for Innovation Questionnaire is known in this case by the acronym TCI (Team Climate Inventory).

The climate questionnaire, as both TSI and TCI, was developed to measure four aspects considered essential for effective team operation and propensity to innovation:

- Participative safety: how much trust that participating team members feel within the group when explaining their opinions and ideas. Accordingly, there can be said to be a teamwork climate with a greater tendency to member participation or Participative climate, as opposed to a Directed work climate.
- Support for innovation: support provided by the team for innovative ideas. So, a teamwork climate can be defined as tending towards innovation or as an Innovative climate, as opposed to a Conservative work climate.
- Team vision: how clearly the team defines goals. Therefore, a teamwork climate can be said to tend towards taking a team view or Unanimous climate, as opposed to a Compartmentalised climate.
- Task orientation: how much effort the team puts into achieving excellence in what it does. Accordingly, a teamwork climate can be considered to be demanding and committed to quality or Climate for Quality and Excellence, as opposed to a Conformist Climate.

The aspects described above and their possible values are set out in Table 1 for each of the three components (People, Task and Team Behaviour) of the development process. The aspects can then be decomposed into subaspects. The aspects referring to the task features and the real teamwork climate will be used to characterise the two ways of developing software now in use: agile and heavy weight processes (section 6). All the aspects in Table 1 are being measured to determine what impact they have on product quality and team productivity, as well as on the satisfaction of the team members. In other words, these aspects have been included in the cross-experiment described in section 6 to empirically study what types of people and people combinations (teams) provide a better fit for each software process type.

4.4. Outputs

Team performance refers to the extent to which team goals, such as productivity, delivery time and product quality or services, are achieved. According to Hackman and Oldham (1980), performance is also related to aspects that are not explicitly part of the team goals, such as job satisfaction or team feasibility. The factors that are usually measured and assessed in the final result are: satisfaction and performance. There are two sides to team effectiveness:

- Explicit aspects or formal performance, which refer to how the team goals are achieved (to what extent, delivery time and product and services quality).
- Non-explicit aspects, which are related to other points, such as job satisfaction and team feasibility.

Effectiveness is manifest at both group and member level from the viewpoint of satisfaction. Satisfaction mainly reflects the affective side of personal results. At work team level, it refers to fulfilling social needs, as well as the aspiration to belong to the group.

COMPONENT	ASPECT/ SUBASPECT	VALUES
PEOPLE	Personality (Big Five)	- Extraversion - Agreeableness - Conscientiousness - Emotional Stability - Openness to Experience
	KSA	- Knowledge - Skills - Abilities
	Climate (Preferences or TSI) • Participative safety • Support for innovation • Team vision • Task orientation	- Participative / Directed - Innovative / Conservative - Unanimous / Compartmentalised - Quality / Conformist
TASK	Autonomy	- Autonomous/ Reliant
	Interdependency between team tasks	- Interdependence / Independence
	Task routineness	- Routine/Creative
TEAM BEHAVIOUR	Cohesion	- Cohesive / Cooperative
	Conflict	- Compulsion / Negotiation
	Climate (Perceptions or TCI) • Participative safety • Support for innovation • Team vision • Task orientation	- Participative / Directed - Innovative / Conservative - Unanimous / Compartmentalised - Quality / Conformist

Table 1. Components, aspects and values considered in software development team forming

5. SOFTWARE DEVELOPMENT TEAM BEHAVIOUR RELATED WORK

There has been little research into group aspects applied to software development. There are studies that use a standard test, like the "Myers-Briggs Type Indicator" (Rutherford, 2001; Bostrom & Kaiser, 1981; Teague, 1998; Hardiman, 1997), to determine the guidelines for team success according to software engineer personality types. There is another study that determines the connection between abilities and personality traits and team performance (White & Leifer, 1986). These studies omit KSA from the individual factors and confidence, conflict, climate, etc., from the group factors.

There are also quantitative team forming methods based on a quantitative abilities model (Burdett & Li, 1995), but they do not consider the softer aspects, such as team member personality factors and actual team personality.

Another team forming method is based on analysing re quired and available skills (Zakarian & Kusiak, 1999), but it does not indicate how to evaluate people's skills. Neither does it consider task and group characteristics.

Zuser and Grechening (2003) propose the use of a questionnaire based on abilities and personality traits that provide the team with information during development on finished software projects to improve team performance. The team is built according to the team forming phases of Tuckman's model: forming, storming, norming, performing and adjourning (Tuckman, 1965). This study combines individual personality and group factors only, and omits KSAs and task characteristics.

We find then that there is no study that covers all the aspects that social psychology has shown to be relevant for software development team forming. Neither do the analysed studies take into account climate, which is a factor that is involved in effective and efficient software development team forming.

6. CHARACTERISING DIFFERENT DEVELOPMENT STRATEGIES BASED ON TEAM BEHAVIOUR

There are now two opposing philosophies on the software process and, therefore, on how to work and cooperate during development: agile methods (Beck et al., 2001; Fowler, 2001; Jeffries et al., 2001) and the heavy-weight or traditional software development process models (IEEE, 1997; ISO/IEC, 2002). Agile methods are put forward as people-oriented adaptable models, whereas the traditional approaches are activity- and product-driven.

It is to be expected that neither approach is better than the other in absolute terms, but one will outperform the other under certain circumstances. These process types are not exclusive, and there is a spectrum of process types, ranging from the pure heavy-weight models, through mixed heavy-weight (for example, small documentation and roles hierarchy preestablished by the project manager), mixed light-weight (decision-making participation but more demanding documentation) processes, to pure agile methods.

Psychological studies of Structural Contingency Theory defend that team performance very much depends on the task. However, the development task features will vary depending on whether an agile or a heavy-weight process is in place. For example, processes application and people assignation, documentation, etc., are more Reliant in a heavy-weight process and more Autonomous for agile methods.

We want to empirically study what type of people, people combinations and team behaviours at climate level better match each software process. For this purpose, first we describe the heavy-weight and agile methods according to particular development features (Table 2). Second, considering the descriptions for each of these development features (Table 2), we characterise the two process types according to the psychological team behaviour criteria discussed earlier. In particular, we consider the aspects: Interdependence, Autonomy and Routineness for the Task component and Cohesiveness, Conflict and real teamwork Climate (TCI) within the Team Behaviour component (specified in Table 1). Table 3 shows the values of the aspects analysed for the agile and heavy-weight process models.

DEVELOPMENT FEATURES	AGILE METHODS (Beck, 1999; Beck et al., 2001; Fowler, 2001; Jeffries et al., 2001)	HEAVY-WEIGHT METHODS (IEEE, 1997; ISO/IEC, 2002)
REQUIREMENTS MANAGEMENT	Short requirements (story cards) are written for a product feature or characteristic. They have a 10-to-20 day term, that is, the scope of the requirements will be developed within this time period. Story cards are used to estimate priorities, scope and development time.	They are based on the elicitation and in-depth analysis of the user requirements from which the functional and non-functional system requirements are defined.
DEVELOPMENT MODEL	Non-systematic development model, sharing an incremental organisational model, based on small deliveries with short cycles. Development by short iterations. Short-term planning for each iteration.	Systematic development model, in which project development is ordered organisationally: estimation, planning, analysis, design, implementation, verification and validation, installation. More orderly development, where iterations are more comprehensive and usually affect more than one stage. Medium- and long-term planning, normally covering the whole project and gradually adjusted as the project progresses.
CHANGE MANAGEMENT	Adaptable, acceptation of change as an inherent part of the development process.	Predictive, where changes should be evaluated to determine whether are not they are addressed in this development process or future versions.
CUSTOMER RELATIONS	The client works with the development team and in the same physical space throughout the whole project. The client closely cooperates in the project. The client gets a better view of the real project status.	The client and development team have a contractual relationship. The client or user participates in defining system requirements and the acceptance of some of the project stages. However, the real project status is unknown to the client.
PEOPLE ASSIGNATION	People-oriented process. People are not considered replaceable parts. Support should be given to the actual members of the development team and not the generic process roles. As it is an adaptable process, a very effective team of creative, highly qualified developers who are good a communicating, highly sociable and highly adaptable to changes and work well as a team is needed. Self-managed or autonomous teams decide on their own technical work and its planning.	Task- or activity-oriented process. The important thing is the activity under development, not who is doing it. People should rotate to prevent any delays caused by new members joining. People are allocated to particular roles and are not usually interchangeable. A good understanding between team members is also important.
PROCESS APPLICATION	The processes are accepted, the whole team should be actively committed to or involved in applying the processes. This means that developers and management are in the same decision-making position.	The processes are usually established, normally by the management, disregarding the opinion and appraisal of development team members, as well as the fitness to the project type under development.
DOCUMENTATION	Fewer management activities. Development is directly code driven, suggesting that the important part of the documentation is the source code and demanding a minimum and little documentation for a particular activity.	More management activities. More bureaucracy. Much more extensive and specific documentation for each task or development activity.

Table 2. Description of agile and heavy-weight description according to software development features

DEVELOPMENT FEATURES	METHOD	TASK	COHESION AND CONFLICT	CLIMATE (TCI)
REQUIREMENTS MANAGEMENT	Agile	Independence	Cohesive &Negotiation	Participative
	Heavy-weight	Interdependence	Cooperative & Negotiation	Directed
DEVELOPMENT MODEL	Agile	Independence (continuous)	Cohesive & Negotiation	Unanimous
	Heavy-weight	Interdependence (stagewise)	Cooperative & Compulsion	Compartmentalised
CHANGE MANAGEMENT	Agile	Autonomous (adaptable)	Negotiation	Innovative
	Heavy-weight	Reliant (structured)	Compulsion	Conservative (accommodating)
CUSTOMER RELATIONS	Agile	Interdependence & Creative	Cohesive & Negotiation	Cohesive
	Heavy-weight	Independence & Routine (planned)	Cooperative & Negotiation	Compartmentalised
PEOPLE ASSIGNATION	Agile	Autonomous (multifunctional)	Cohesive & Negotiation	Participative, Innovative, Unanimous & Quality
	Heavy-weight	Reliant (hierarchical)	Cooperative & Compulsion	Directed, Conservative, Compartmentalised & Quality
PROCESS ASSIGNATION	Agile	Autonomous	Cohesive	Participative & Innovative
	Heavy-weight	Reliant	Cooperative	Directed & Conservative
DOCUMENTATION	Agile	Autonomous	Cooperative	Participative
	Heavy-weight	Reliant	Cohesive	Directed, Conservative & Quality

Table 3. Appraisal of agile and heavy-weight methods by team behaviour

After analysing how development teams are organised in software processes and examining the team behaviour aspects from social psychology, which are considered important in forming development teams to get better performance, we considered collecting data to analyse and advance software engineering at three Spanish universities. There are basically two reasons for doing this: a) it is not intrusive and b) it is not costly.

The computing students are taking different subjects, all related to one of the stages that take place in a software project or with the full software project. However, not all the projects are developed according to the same development philosophy, that is, some projects follow pure or mixed heavy-weight processes, whereas other perform agile or mixed agile processes.

Many aspects are considered to have an influence on team performance, effectiveness and satisfaction. However, we

have taken into account: the personality of each team member; knowledge, skills and abilities on how to do teamwork, preferences and perceptions of the teamwork environment or climate in which the software project is developed, and, finally, the level of cohesion and conflict achieved by the team. These aspects have been selected, because Yang & Tang (2004) consider them to have a significant impact on the above-mentioned team response variables, where, however, they were measured in separate ways rather than as an integral approach as here. From these aspects and their relationships measured in the cross-experiment, we will be able to find out what type of people/people combinations (team) are better for the different software development types.

They are all aspects examined in the field of social psychology. Therefore, to be able to determine how much each of these aspects is involved in the results achieved by the team, we are taking measurements on each one using questionnaires prepared to get findings on how to relate people individually and at team level depending on the software process model, as well as to see the results on the product (quality and development time) and the people (job satisfaction within the team). For this purpose, specialists in social psychology are running a cross-experiment on the students, which, additionally, are helping in data analysis, as well as their interpretation:

The cross-experiment is divided into three parts:

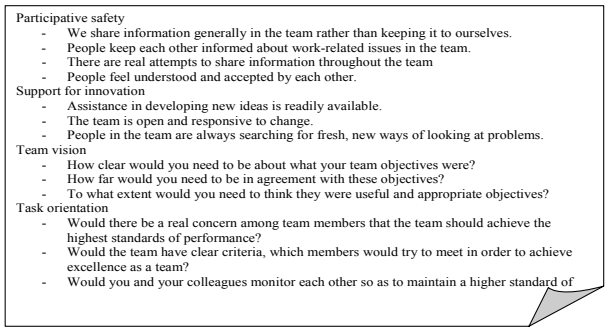
1. Preliminary phase or PRE
2. Development phase or DURING
3. Final phase or POST

and the questionnaires will be designed according to this division, as set out in Table 4. A detail of the items of the team climate inventory (TCI) questionnaire (Anderson & West, 1998; Anderson & West, 1999) used to measure the real team climate in development teams is shown in Figure 2.

	CROSS-EXPERIMENT PHASES		
MEASURED ASPECTS	PRE	DURING	POST
Personality(Big Five)	X		
KSA	X		
Climate (TSI)	X		
Climate (TCI)		X	X
Interdependence, Autonomy and Routineness			X
Conflict and Cohesion		X	

Table 4. Cross-experiment phases and aspects

All the measures that are taken from the start to the end of the cross-experiment are independent variables, except the level of Satisfaction, which like Performance, is a dependent or response variable. Performance includes the development time (measured as a function of software products delivery date compliance) and product quality (defined by the criteria of document evaluation). Team member satisfaction at the end of the project is measured by means of a special-purpose questionnaire used in social psychology.


Participative safety
- We share information generally in the team rather than keeping it to ourselves.
- People keep each other informed about work-related issues in the team.
- There are real attempts to share information throughout the team
- People feel understood and accepted by each other.

Support for innovation
- Assistance in developing new ideas is readily available.
- The team is open and responsive to change.
- People in the team are always searching for fresh, new ways of looking at problems.

Team vision
- How clear would you need to be about what your team objectives were?
- How far would you need to be in agreement with these objectives?
- To what extent would you need to think they were useful and appropriate objectives?

Task orientation
- Would there be a real concern among team members that the team should achieve the highest standards of performance?
- Would the team have clear criteria, which members would try to meet in order to achieve excellence as a team?
- Would you and your colleagues monitor each other so as to maintain a higher standard of

Figure 2. Detail of the team climate inventory (TCI) questionnaire

The aim is to cover the range of possible processes from pure heavy-weight to pure agile processes. What we are relating for each of these development types are people aspects (personality, KSA and climate preferences) along with the combinations of people in each team and the product quality, development time and team member satisfaction. The types of things we expect to find is how one type of person behaves individually and at team level in an agile process and determine the person/team behaviour match to each software process.

7. CONCLUSIONS

Social psychology has not yet managed to come up with an answer to the question of how to form teams to optimise the final result. Additionally, the research carried out in social psychology on the subject of team performance is specialised in a field or task type. This means that to gain knowledge about software development team forming, we need to conduct specialised studies in our area.

This is precisely the goal of the cross-experiment that we are conducting. Measures are taken through the questionnaires on aspects and subaspects that affect each team behaviour model component (Table 1). This way we can characterise people according to personality factors (Big Five) and KSAs, as well as team behaviour through two climate levels, preferences (TSI) and perceptions (TCI). On the other hand, we characterise the different software process models, agile and heavy-weight, considering the extreme cases, pure agile and pure heavy weight processes, although the cross-experiment is also planned on projects that lie somewhere in-between (mixed heavy-weight and mixed light-weight methods). The software processes are characterised considering particular features of the task (interdependence, autonomy and routineness) and the aspects that have an impact on the team behaviour component (cohesion, conflict and climate perceived in the team (TCI)), matching their possible values to software process features, agile and heavy weight (Table 2 and Table 3). Having completed these characterisations, we look at what

type of people, both individually (personality factors gathered using the Big Five) and at team level (climate preferences (TSI)) better fit in with either software process type and, therefore, determine the person/team behaviour match for each software process. The cross-experiment is now yielding preliminary findings, and we see how sure and innovative people with the ability to reason, synthesise and work together as a team and who prefer a unanimous and innovative climate perform better in agile processes when the team is composed of 66% of such members and the other 44% are diligent people with knowledge of teamwork but who prefer a compartmentalised climate. On the other hand, teams with diligent and sure people who are good at teamwork if reserved for given activities and prefer a comfortable, compartmentalised, quality climate perform better in heavy-weight processes when the team has 53% of such members and the remainder of the membership are innovative people who prefer a cohesive and participative climate.

REFERENCES

Acuña ST, Juristo N. Assigning people to roles in software projects. *Software: Practice and Experience* 2004; 34:675-696.

Anderson N, Burch GJ. *The Team Selection Inventory*. ASE. NFER-Nelson, Slough, 2003.

Anderson N, West M. *The Team Climate Inventory*. ASE. NFER-Nelson, Windsor, 1994.

Anderson N, West M. Measuring climate for work group innovation: Development and validation of the team climate inventory. *Journal of Organizational Behaviour* 1998; 19:235-258.

Anderson N, West M. *The Team Climate Inventory: User's Guide,* 2nd ed. ASE. NFER-Nelson, Windsor, 1999.

Barrick MR, Mount MK. The Big Five personality dimensions and job performance: A meta-analysis. *Personnel Psychology* 1991; 44:1-26.

Barry B, Stewart GL. Composition, process and performance in self-managed groups: The role of personality. *Journal of Applied Psychology* 1997; 82:62-78.

Beck K. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, Reading, 1999.

Beck K, Beedle M, Cockburn A, Cunnimgham W, Fowler M et al., *Agile Manifesto*, http://agilemanifesto.org/, 2001.

Boehm BW, Abts C, Brown WA, Chulani S, Clark BK, Horowitz E, Madachy R, Reifer DJ, Steece B. *Software Cost Estimation with COCOMO II.* Upper Saddle River: Prentice Hall PTR, 2000.

Bostrom RP, Kaiser KM. Personality differences within systems project teams: Implications for designing solving centers. *Proceedings of the Eighteenth Annual ACM SIGCPR Conference* 1981; 248–285.

Breaugh JA. The measurement of work autonomy. *Human Relations* 1985; 38:551-570.

Burch GJ, Anderson N. Measuring person-team fit: Development and validation of the team selection inventory. *Journal of Managerial Psychology* 2004; 19(4):406-426.

Burdett G, Li R-Y. A quantitative approach to the formation of workgroups. *Proceedings of the ACM SIGCPR Conference* 1995; 202–212.

Curtis B, Hefley WE, Miller S. People Capability Maturity Model (P-CMM) Version 2.0. *Maturity Model CMU/SEI-2001-MM-001*. Carnegie Mellon University, Software Engineering Institute, 2001.

DeMarco T, Lister T. *Peopleware: Productive Projects and Teams* (2nd ed). Dorset House, New York, 1999.

Digman JM. Personality structure: Emergence of the five-factor model. *Annual Review of Psychology* 1990; 41:417-440.

Evans BK, Fischer DG. A hierarchical model of participatory decision-making, job autonomy, and perceived control. *Human Relations* 1992; 45:1169-1189.

Fowler M. Is design dead? *Proceedings of the XP2000*, http://www.martinfowler.com/articles/designDead.html, http://www.refactoring.com/, 2001.

Hackman JR, Oldham GR. *Work Redesign*. Addison-Wesley, Reading, 1980.

Hardiman LT. Personality types and software engineers. *IEEE Computer* 1997; 301(10):10–10.

Hendriks AA. *The construction of Five-Factor Personality Inventory (FFPI)*, Dissertation, University of Groningen, Groningen, The Netherlands, 1997.

Hollenbeck JR, Moon H, Ellis APJ, West BJ, Ilgen DR, Sheppard L, Porter COLH, Wagner III JA. Structural contingency theory and individual differences: examination of external and internal person-team fit. *Journal of Applied Psychology* 2002; 87(3):599-606.

IEEE Standard for Developing Software Life Cycle Processes, IEEE Standard 1074-1997.

ISO/IEC International Standard: Information Technology. Software Life Cycle Processes, Amendment 1, ISO/IEC Standard 12207-1995/Amd. 1-2002.

Jeffries R, Anderson A, Hendrikson C. *Extreme Programming Installed*. Addison-Wesley, 2001.

Katzenbach J, Smith D. *The Discipline of Teams: A Mindbook-workbook for Delivering Small Group Performance.* John Wiley & Sons, 2001.

Kellner MI, Madachy RJ, Raffo DM. Software Process Simulation Modelling: Why? What? How? *Journal of Systems and Software* 1999; 46:91-105.

Koontz H, O'Donnell C. *Principles of Management: An Analysis of Managerial Functions*. McGraw-Hill, 1972.

Larson C, LaFasto FMJ. *When Teams Work Best: 6.000 Team members and leaders tell what it takes to excel.* Sage Publications, 2001.

McGrath JE. *Groups: Interaction and Performance*. Prentice-Hall, Englewood Cliffs, NJ, 1984.

Molleman E, The modalities of self-management: the "must", "may", "can" and "will" of local decision making, *The International Journal of Operations and Production Management* 2000; 20.

Molleman E, Nauta A, Jehn KA. Person-Job Fit Applied to teamwork: A Multi-Level Approach. *Proceeding of the 6th International Workshop on Teamworking, Scholl of Technology and Society* 2002.

Molleman E, Slomp J. Functional flexibility and team performance. *International Journal of Production Research* 1999; 37:1837-1858.

Moore E. Personality characteristics of information systems professionals. *Proceedings of the Conference on SIGCPR* 1991; 140–155.

Rutherfoord RH. *Using personality inventories to help form teams for software engineering class projects*. *SIGCSE-Bulletin* 2001; 33(3):76–76.

Steiner ID. *Group process and productivity,* New York: Academic Press, 1972.

Teague J. Personality type, career preference and implications for computer science recruitment and teaching. *Proceedings of the Third Australasian Conference on Computer Science Education* 1998; 155–63.

Tuckman B. Developmental sequence in small groups. *Psychological Bulletin* 1965; 63:384–399.

Turley R, Bieman J. Competencies of exceptional and nonexceptional software engineers. *The Journal of Systems and Software* 1995; 28(1):19–38.

Van den Beukel AL, Molleman E. Downsides of multifunctionality in team-based-based work. *Personnel Review* 2002; 31:482-494.

West MA. The social psychology of innovation in groups. In M. A. West, J. L. Farr (Eds.), *Innovation and Creativity at Work*. Wiley, 1990.

West MA, Anderson N. Innovation in top management teams. *Journal of Applied Psychology* 1996; 81:680-693.

White K, Leifer R. Information systems development success: Perspectives from project team participants. *MIS Quarterly* 1986; 10(3):215–23.

Wynekoop J, Walz D. Investigating traits of top performing software developers. *Information Technology & People* 2000; 13(3):186-195.

Yang, H-L, Tang J-H. Team structure and team performance in IS development: a social network perspective. *Information & Management* 2004; 41:335-349.

Zakarian A, Kusiak A. forming teams: An analytical approach. *IIE Transactions* 1999; 31:85–97.

Zuser W, Grechening T. Reflecting skills and personality internally as means for team performance improvement. *Proceedings of the 16th Conference on Software Engineering Education and Training, IEEE Computer Society*, 2003.

Section II

Process Modeling – Focus on Model Implementation

Modeling Recruitment and Role Migration Processes in OSSD Projects

Chris Jensen and Walt Scacchi
Institute for Software Research
Bren School of Information and Computer Sciences
University of California, Irvine
Irvine, CA USA 92697-3425
{cjensen, wscacchi}@ics.uci.edu

Abstract

Socio-technical processes have come to the forefront of recent analyses of the open source software development (OSSD) world. Though there many anecdotal accounts of these processes, such narratives lack the precision of more formal modeling techniques, which are needed if these processes are going to be systematically analyzed, simulated, or re-enacted. Interest in making these processes explicit is mounting, both from the commercial side of the industry, as well as among spectators who may become contributors to OSSD organization. Thus, the work we will discuss in this paper serves to close this gap by analyzing and modeling recruitment and role transition processes across three prominent OSSD communities whose software development processes we've previously examined: Mozilla.org, the Apache community, and NetBeans.

Keywords: Project recruitment, membership, process modeling, open source, Mozilla, Apache, NetBeans

Introduction

In recent years, organizations producing both open and closed software have sought to capitalize on the perceived benefits of open source software development (OSSD) methodologies. This necessitates examining the culture of prominent project communities in search of ways of motivating developers. Although the ensuing studies have provided much insight into OSSD culture, missing from this picture was the process context that produced the successes being observed. Ye and Kishida (2003) and Crowston and Howison (2005) observe that community members gravitate towards central roles over time represented with "onion" diagrams such as in figure 1. These depictions indicate a similar number of layers in organizational hierarchies across communities, but do not suggest how one might transition between layers and what roles are available at each layer. Much like their development processes, OSSD communities typically provide little insight into role migration

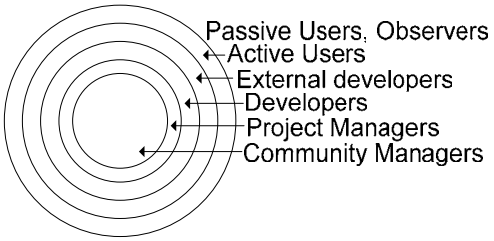


Figure 1. An "onion" diagram representation of an open source community organizational hierarchy

processes. What guidance is provided is often directed at recruitment- initial steps to get people in the door. Guidance for attaining more central roles is often characterized as being meritocratic, depending on the governance structure of the community. Nevertheless, these development roles and how developers move between them seems to lie outside of the traditional view of software engineering, where developers seem to be limited to roles like requirements analyst, software designer, programmer, or code tester, and where there is little/no movement between roles (except perhaps in small projects).

Christie and Staley (2000) argue that social and organizational processes, such as those associated with moving between different developer roles in a project, are important in determining the outcome of software development processes. In previous studies, we have examined software development processes within and across OSSD communities (Jensen and Scacchi, 2005, Scacchi 2002, 2004, 2005). Here, we take a look at two related socio-technical processes used in OSSD as a way of merging the social/cultural and technical/developmental OSSD activities. Specifically, we'll focus on the recruitment and migration of developers from end-users or infrequent contributors towards roles more central to the community, like core developer, within projects such as the Mozilla, Apache community, and NetBeans projects. Such processes characterize both the hierarchy of roles that OSS developers play (cf. Gacek and Arief 2004), as well as how developers move through or become upwardly mobile within an OSSD project (Sim and Holt 1998). While anecdotal evidence of these

processes exists, the lack of precision in their description serves as a barrier to community entry, continuous improvement, and process adoption by other organizations. The goal of our work here thus serves to provide process transparency through explicit modeling of such processes in ways that may enable increased community participation, more widespread process adoption, and process improvement.

In the remaining sections, we outline details aboutrecruitment and role migration as membership processes as found while examining each of these three OSSD project communities. At the ProSim'05 Workshop we will present a variety of semi-structured and formal models that enable more rigorous analysis and simulated re-enactment using tools and techniques we have previously developed and employed (cf. Noll and Scacchi 2001, Jensen and Scacchi 2005)

Membership Processes in Mozilla.org

Developer recruitment in Mozilla was difficult at the start. The opening of the Netscape browser source code offered developers a unique opportunity to peek under the hood of the once most dominant Web browser in use. Nevertheless, the large scale of the application (multi-million source lines of code) and the complex/convoluted architecture scared developers away. These factors, combined with the lack of a working release and the lack of support from Netscape led one project manager to quit early on (Mockus, *et. al,* 2002). However, with the eventual release of a working product, the Mozilla project garnered users who would become developers to further the cause.

The Mozilla Web site lists several ways for potential developers and non-technical people to get involved with the community (Getting Involved with Mozilla.org, 2005). The focus on quality assurance and documentation reflects a community focus on maturing, synchronizing, and stabilizing updates to the source code base. Technical membership roles and responsibilities currently listed include bug reporting, screening, confirming, and fixing, writing documentation, and contacting sites that do not display properly under Mozilla. Compared to more central roles, these activities do not require deep knowledge of the Mozilla source code or system architecture, and serve to allow would-be contributors to get involved and participate in the overall software development process.

When bugs are submitted to the Bugzilla, they are initially assigned to a default developer for correction. It is not uncommon for community developers and would-be developers to become frustrated with an outstanding issue within the bug repository and submit a patch, themselves.

The next task is to recruit others to accept the patch and incorporate it into the source tree. Recruitment of patch review is best achieved through emailing reviewers working on the module for which the patch was committed or reaching out to the community via the Mozilla IRC chat. By repeatedly demonstrating competency and dedication writing useful code within a section of the source, would-be developers gain a reputation among those with commit access to the current source code build tree. Eventually, these committers recommend that the developer be granted access by the project drivers. In rare cases, such a developer may even be offered ownership of a particular module if s/he is the primary developer of that module and it has not been blocked for inclusion into the trunk of the source tree[1].

Once a project contributor is approved as a source code contributor, there are several roles available to community members. Most of these are positions requiring greater seniority or record of demonstrated accomplishments within the community. As module developers and owners establish themselves as prominent community members, other opportunities may open up. In meritocratic fashion (cf. Fielding 1999), developers may transition from being a QA module contact to a QA owner. Similar occasions exist on the project level for becoming a module source reviewer.

Super-reviewers attain rank by demonstrating superior faculty for discerning quality and effect of a given section of source on the remainder of the source tree. If a reviewer believes that s/he has done this appropriately, s/he must convince an existing super-reviewer of such an accomplishment. This super-reviewer will propose the candidate to the remainder of the super-reviewers. Upon group consensus, the higher rank is bestowed on the reviewer (Mozilla Code Review FAQ, 2005). The same follows for Mozilla drivers, who determine the technical direction of the project per release.

Community level roles include smoke-test coordinator, code sheriff, and build engineer, although no process is prescribed for such transitions. As individual roles, they are held until vacated, at which time, the position is filled by appointment from the senior community members and Mozilla Foundation staff. Role hierarchy and a flow graph of the migration process for transitioning from reviewer to super-reviewer are provided in figure 2 as an example of those we

[1] https://bugzilla.mozilla.org/show_bug.cgi?id=18574

have modeled for this community. In the flow graph, rectangles refer to actions, whereas ovals refer to resources created or consumed by the associated action, as determined by the direction of the arrow linking the two. Transitions from one role to another are depicted with a dashed arrow from an action performed by one role to the title of another. We have also used dashed lines to differentiate social or role transitioning activities and resources from strictly technical, developmental resources.

Membership Processes in the Apache Community

Role migration in the Apache community is linear. The Apache Software Foundation (ASF) has laid out a clear path for involvement in their meritocracy. Individuals start out as end-users (e.g., Web site administrators), proceed to developer status, then committer status, project management committee (PMC) status, ASF membership, and lastly, ASF board of directors membership (How the ASF Works, 2005). Much as in advancement in the Mozilla community, Apache membership is by invitation only. As the name suggests, the Apache server is comprised of patches submitted by developers. These patches are reviewed by committers and either accepted or rejected into the source tree.

In addition to feature patches, developers are also encouraged to submit defect reports, project documentation, and participate on the developer mailing lists. When the PMC committee is satisfied with the developer's contributions, they may elect to extend an offer of "committership" to the developer, granting him/her write access to the source tree. To accept committership, the developer must submit a contributor license agreement, granting the ASF license to the intellectual property conveyed in the committed software artifacts.

PMC membership is granted by the ASF. To become a PMC member, the developer/committer must be nominated by an existing ASF member and accepted by a majority vote of the ASF membership participating in the election (Fielding, et. al, 2002). Developers and committers nominated to become PMC members have demonstrated commitment to the project, good judgment in their contributions to the source tree, and capability in collaborating with other developers on the project. The PMC is responsible for the management of each project within the Apache community. The chair of the PMC is an ASF member elected by his/her fellow ASF members who initially organizes the day-to-day management infrastructure for each project, and is ultimately responsible for the project thereafter. ASF membership follows the same process as PMC membership- nomination and election by a majority vote of existing ASF members.

ASF members may run for office on the ASF board of directors, as outlined by the ASF bylaws (Bylaws of the Apache Software Foundation, 2005). Accordingly, the offices of chairman, vice chairman, president, vice president, treasurer (and assistant), and secretary (and assistant) are elected annually. A flow graph of the role migration process appears in figure 3.

Although, there is one path of advancement in the Apache community, there are several less formal committees that exist on a community (as opposed to project) scale. These include the conference organizing committee, the security committee, the public relations committee, the Java Community Process (JCP) committee, and the licensing committee. Participation in these committees is open to all committers (and higher ranked members) and roles are formalized on an as-needed basis (e.g. conference organization). Non-committers may apply for inclusion in specific discussion lists by sending an email to the board mailing alias explaining why access should be granted. Thus, processes associated with these committees are ad hoc and consist of one step.

Membership Processes in the NetBeans.org Community

Roles in the NetBeans.org community for developing the Java-based NetBeans interactive development environment are observable on five levels of project management (Oza, et. al 2002) just as in Apache. These range from users to source contributors, module-level managers, project-level managers, and community-level managers. The NetBeans community's core members are mostly Sun Microsystems employees, the community's primary sponsor, and are subject to the responsibilities set on them by their internal organizational hierarchy. As such, (and unlike the cases of Apache and Mozilla), not all roles are open to volunteer and third-party contributors. Non-Sun employed community members wanting to participate beyond end-usage are advised to start out with activities such as quality assurance (QA), internationalization, submitting patches, and documentation (Contributing to the NetBeans Project, 2005). As in the case with Mozilla, until they have proven themselves as responsible, useful, and dedicated contributors, developers must submit their contributions to developer mailing lists and the issue repository, relying on others with access to commit the source. However, unlike Mozilla, developers are also encouraged to start new modules.

While the community was more liberal with module creation early in the project's history, as the community has matured, additions to the module catalogue have become more managed to eliminate an abundance of abandoned modules. Also as in Mozilla, developers are subjected to the proving themselves before being granted committer status on a portion of the source tree. Additionally, they may gain module owner status be creating a module or taking over ownership of an abandoned module that they have been the primary committer for. With module ownership comes the responsibility to petition the CVS manager to grant commit access to the source tree to developers, thereby raising their role status to "committer."

Rising up to the project-level roles, the Sun-appointed CVS source code repository manager is responsible for maintaining the integrity of the source tree, as well as granting and removing developer access permissions. In contrast, the release manger's role is to coordinate efforts of module owners to plan and achieve timely release of the software system. Theoretically, any community member may step in at any time and attempt to organize a release. In practice, this rarely occurs. Instead, most community members passively accept the roadmap devised by Sun's NetBeans team. In the latter case, the previous release manager puts out a call to the community to solicit volunteers for the position for the upcoming cycle. Assuming there are no objections, the (usually veteran) community member's candidacy is accepted and the CVS manager prepares the source tree and provides the new release manager permissions accordingly. Alternatively, a member of Sun may appoint a member of their development team to head up the release of their next development milestone.

At the community-management level, the community managers coordinate efforts between developers and ensures that issues brought up on mailing lists are addressed fairly. At the inception of the NetBeans project, an employee of CollabNet (the company hosting the NetBeans Web portal) originally acted as community manager and liaison between CollabNet and NetBeans. However, it was soon transferred to a carefully selected Sun employee (by Sun) who has held it since. As community members have risen to more central positions in the NetBeans community, they tend to act similarly, facilitating and mediating mailing list discussions of a technical nature, as well as initiating and participating in discussions of project and community direction.

Lastly, a committee of three community members, whose largely untested responsibility is to ensure fairness within the community, governs the NetBeans project.. One of the three is appointed by Sun. The community at large elects the other two members of the governance board. These elections are held every six months, beginning with a call for nominations by the community management. Those nominees that accept their nomination are compiled into a final list of candidates to be voted on by the community. A model of the product development track role migration process is shown in figure 4.

Discussion

In both NetBeans and Mozilla, recruitment consists of listing ways for users and observers to get involved. Such activities include submitting defect reports, test cases, source code and so forth. These activities require a low degree of interaction with other community members, most notably decision makers at the top of the organizational hierarchy. Our observation has been that the impact of contributions trickles up the organizational hierarchy whereas socio-technical direction decisions are passed down. As such, activities that demonstrate capability in a current role, while also coordinating information between upstream and downstream (with respect to the organizational hierarchy) from a given developer are likely to demonstrate community member capability at his/her current role, and therefore good candidates for additional responsibilities.

Recruitment and role migration processes aren't something new; since they describe the actions and transition passages involved in moving along career paths. Like career paths described in management literature (e.g., Lash and Sein 1995), movement in the organizational structure may be horizontal or vertical. Most large OSSD project communities are hierarchical, even if here are few layers to the hierarchy and many members exist at each layer.

In the communities we have examined, we found different paths (or tracks) towards the center of the developer role hierarchy as per the focus of each path. Paths we've identified include project management (authority over technical issues) and organizational management (authority over social/infrastructural issues). Within these paths, we see tracks that reflect the different foci in their software processes. These include quality assurance roles, source code creation roles, and source code versioning roles (e.g. cvs manager, cvs committer, etc), as well as role paths for usability, marketing, and licensing. There are roles for upstream development activities (project planning--these are generally taken up by more senior members of the community. This is due in part that developers working in these roles can have an

impact on the system development commensurate with the consequences/costs of failure, and require demonstrated skills to ensure the agents responsible won't put the software source code into a state of disarray).

In comparison to traditional software development organizations, tracks of advancement in open source communities are much more fluid. A developer contributing primarily to source code generation may easily contribute usability or quality assurance test cases and results to their respective community teams. This is not to suggest that a module manager of a branch of source code will automatically and immediately gain core developer privileges, responsibilities, and respect from those teams. However, industrial environments tend towards rigid and static organizational hierarchies with highly controlled growth at each layer.

The depiction of role hierarchies in open source communities as concentric, onion-like circles speaks to the fact that those in the outer periphery have less direct control or knowledge of the community's current state and its social and technical direction compared to those in the inner core circle. Unlike their industrial counterparts, open source community hierarchies are dynamic. Although changes in the number of layers stabilizes early in the community formation, the size of each layer (especially the outer layers) is highly variable. Evolution of the organizational structure may cause or be caused by changes in leadership, control, conflict negotiation, and collaboration in the community, such as those examined elsewhere (Jensen and Scacchi 2005b). If too pronounced, these changes can lead to breakdowns of the technical processes.

As a general principle, meritocratic role migration processes such as those we have observed consist of a sequence of establishing a record of contribution in technical processes in collaboration with other community members, followed by certain "rights of passage" specific to each community. For Apache, there is a formal voting process that precedes advancement. However, in the Mozilla and NetBeans communities, these are less formal. The candidate petitions the appropriate authorities for advancement or otherwise volunteers to accept responsibility for an activity. These authorities will either accept or deny the inquiry.

Conclusion

Social or organizational processes that affect or constrain the performance of software development processes have had comparatively little investigation. This is partially because some of these processes may be well understood (e.g., project management processes like scheduling or staffing), while others are often treated as "one-off" or *ad hoc* in nature, executing in a variety of ways in each instantiation. The purpose of our examination and modeling study of recruitment and role migration processes is to help reveal how these socio-technical processes are intertwined with conventional software development processes, and thus constrain or enable how software processes are performed in practice. In particular, we have examined and modeled these processes within a sample of three OSSD projects that embed the Web information infrastructure. Lastly, we have shown where and how they interact with existing software development processes found in our project sample.

References

Bylaws of the Apache Software Foundation, available online at http://www.apache.org/foundation/bylaws.html accessed 7 February 2005

Christie, A. and Staley, M. "Organizational and Social Simulation of a Software Requirements Development Process" *Software Process Improvement and Practice* 2000; 5: 103–110 (2000)

Contributing to the NetBeans Project, available online at http://www.netbeans.org/community/contribute/ accessed 7 February 2005

Coward, Anonymous. "About Firefox and Mozilla" Comment on Slashdot.org forum "Firefox Developer on Recruitment Policy," available online at http://developers.slashdot.org/comments.pl?sid=137815&threshold=1&commentsort=0&tid=154&tid=8&mode=thread&cid=11527647, 31 January, 2005.

Crowston, K. and Howison, J. 2005. The Social Structure of Free and Open Source Software Development, First Monday, 10(2). February. Online at http://firstmonday.org/i8ssues/issue10_2/crowston/index.html

Elliott, M., The Virtual Organizational Culture of a Free Software Development Community, *Proceedings of the Third Workshop on Open Source Software*, Portland, Oregon, May 2003.

Fielding, R., Shared Leadership in the Apache Project. *Communications ACM*, 42(4), 42-43, 1999.

Fielding, R., Hann, I-H., Roberts, J and Sandra Slaughter, S. "Delayed Returns to Open Source Participation: An Empirical Analysis of the Apache HTTP Server Project," Presented at the Conference on Open Source: Economics, Law, and Policy, Toulouse, France June 2002.

Gacek, C. and Arief, B., The Many Meanings of Open Source, *IEEE Software*, 21(1), 34-40, January/February 2004.

Getting Involved with Mozilla.org, Web page available online at http://www.mozilla.org/contribute/ 3 November 2004

How the ASF works, available online at http://www.apache.org/foundation/how-it-works.html, accessed 7 February 2005

Jensen, C. and Scacchi, W., Process Modeling Across the Web Information Infrastructure, *Software Process Improvement and Practice,* to appear, 2005.

Jensen, C. and Scacchi, W. Collaboration, Leadership, Control, and Conflict Negotiation Processes in the NetBeans.org Open Source Software Development Community. working paper, Institute for Software Research, March 2005

Lash, P.B. and Sein, M.K. Career Paths in a Changing IS Environment: A Theoretical Perspective, Proc. SIGCPR 1995, 117-130. Nashville, TN

Mockus, A., Fielding, R., and Herbsleb, J. "Two Case Studies of Open Source Software Development: Apache and Mozilla," ACM Transactions on Software Engineering and Methodology, 11(3):309-346, 2002

Mozilla Code Review FAQ, available online at http://www.mozilla.org/hacking/code-review-faq.html, accessed 7 February 2005

Noll, J. and Scacchi, W., Specifying Process-Oriented Hypertext for Organizational Computing, *J. Network and Computer Applications,* 24(1), 39-61, 2001.

Oza, M. Nistor, E. Hu, X., Jensen, C., Scacchi, W. "A First Look at the NetBeans Requirements and Release Process." June 2002, updated February 2004 available online at http://www.isr.uci.edu/~cjensen/papers/FirstLookNetBeans/.

Scacchi, W., Understanding the Requirements for Developing Open Source Software Systems, *IEE Proceedings--Software*, 149(1), 24-39, February 2002.

Scacchi, W., Free/Open Source Software Development Practices in the Computer Game Community, *IEEE Software*, 21(1), 59-67, January/February 2004.

Scacchi, W., Socio-Technical Interaction Networks in Free/Open Source Software Development Processes, in S.T. Acuña and N. Juristo (eds.), *Software Process Modeling*, 1-27, Springer Science+Business Media Inc., New York, 2005.

Sim, S.E. and Holt, R.C., The Ramp-Up Problem in Software Projects: A Case Study of How Software Immigrants Naturalize, *Proc. 20th Intern. Conf. Software Engineering*, Kyoto, Japan, 361-370, 1998.

Ye, Y. and Kishida, K. Towards an Understanding of the Motivation of Open Source Software Developers, *Proc. 25th Intern. Conf. Software Engineering*, Portland, OR, 419-429, 2003.

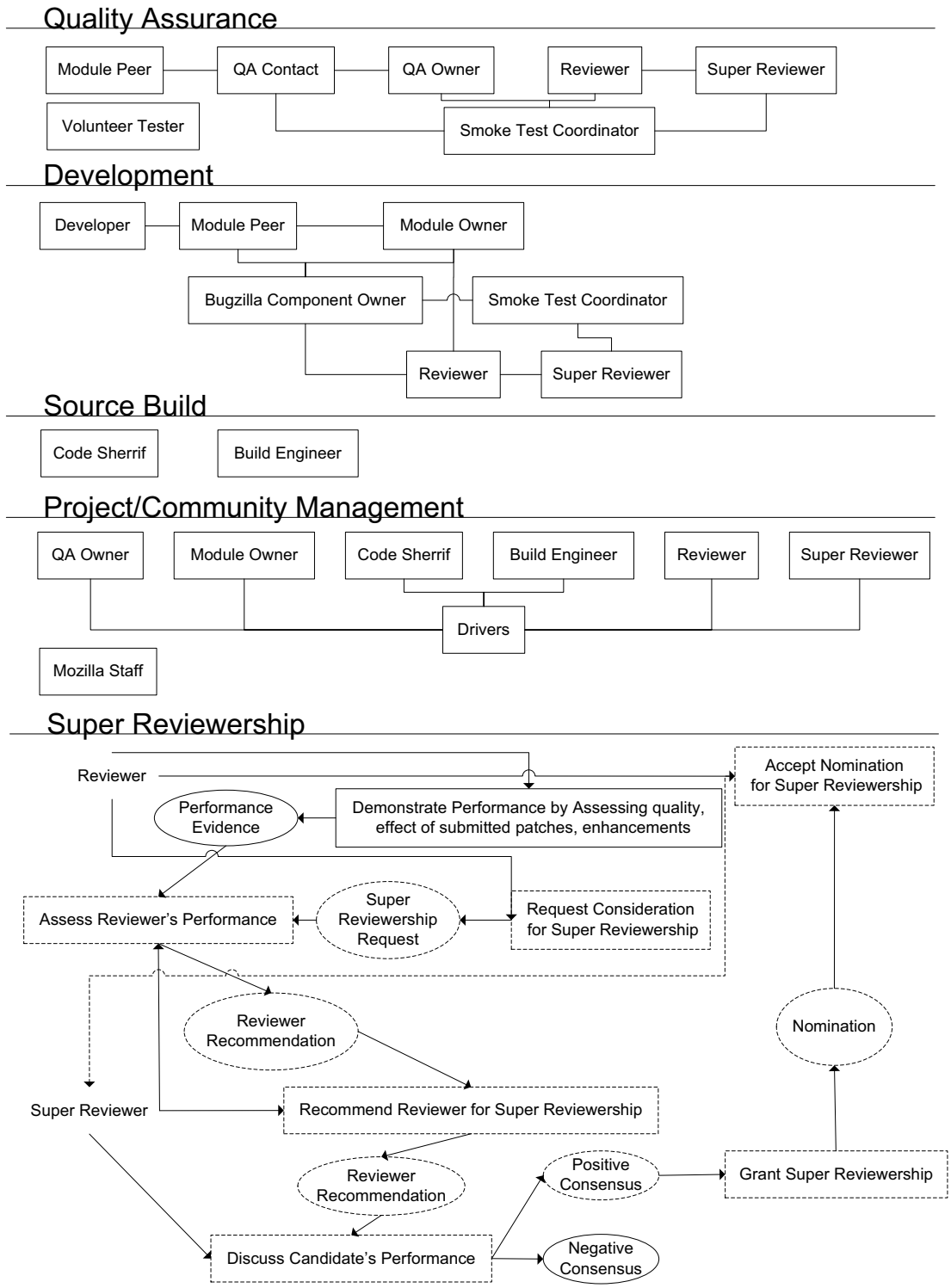


Figure 2. Role hierarchy and super reviewership migration in the Mozilla community

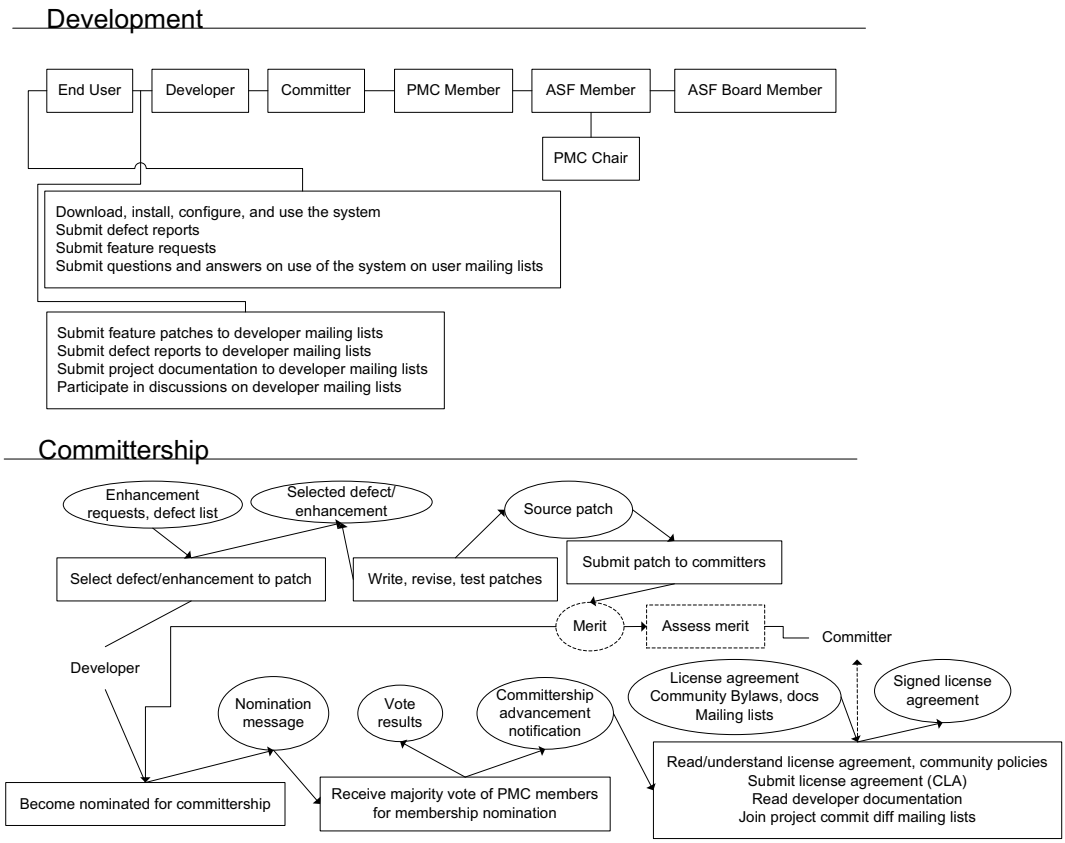


Figure 3. Role hierarchy and committership migration in the Apache community, highlighting the sequence of a developer becoming a committer

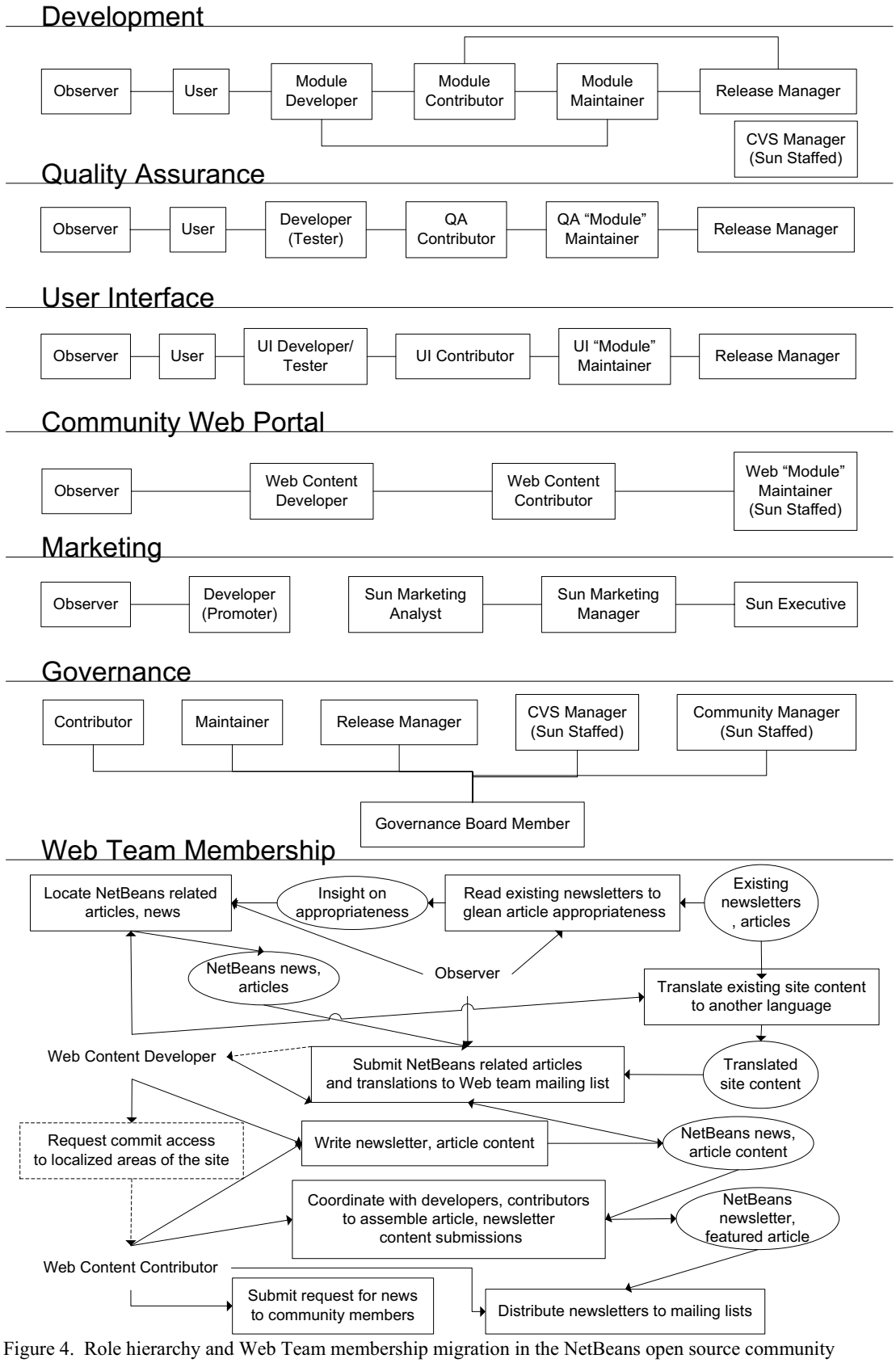


Figure 4. Role hierarchy and Web Team membership migration in the NetBeans open source community

Size Measurement of Embedded Software System Families

Sebastian Kiebusch
Information Systems Institute
Faculty of Economics and Management
University of Leipzig
Germany
kiebusch@wifa.uni-leipzig.de

Bogdan Franczyk
Information Systems Institute
Faculty of Economics and Management
University of Leipzig
Germany
franczyk@wifa.uni-leipzig.de

Andreas Speck
Commercial Information Systems
Faculty of Economics and Business Administration
University of Jena
Germany
andreas.speck@uni-jena.de

Abstract

Embedded software systems have become the driving force in many areas of technology like the automotive industry. Control functions of cars, driver assistance as well as systems for information and entertainment are accomplished by software driven control units. Due to the high complexity and development effort of embedded systems, these resources have to be reused. Software system families are a promising solution to gain a cost reduction by reusing common software assets in different variants of an automobile. To support the economic management of this development approach we need software metrics to estimate the effort of building embedded software system families. Techniques of size measurement and cost estimation for software system families are highly insufficient in general and do not exist for the automotive domain. Therefore, this article describes a conglomerate of innovative metrics to measure the size of a system family orientated software development. These size metrics analyze a real- time and a process focused perspective of embedded software system families in the automotive domain. A combination of both viewpoints describes the unadjusted size of software driven control units to indicate and estimate their development costs.

1. Introduction

Today more than 98% of all microprocessors are used in embedded systems which all are free progammable [4]. Additionally to this broad base we observe an enormous increase of embedded software systems in several domains. The automotive industry is an outstanding sector to visualize this accelerated growth of technology. In these days, up to 90% of new functions and innovations in a car are enabled by embedded software technologies [4].

In contrast to this evolution, in these days not much software is reused but developed in a proprietary way for each type of car again.

First positive reuse concepts may be found in the application object technologies or concepts such as frameworks or components [18]. However, the large number of software systems with large variability and different variants demands techniques for reuse like the concept of software system families (SSF) [4]. Within this framework a SSF is a "… set of software- intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets" [5].

To improve the possibilities of reusing software resources we develop the approach of *Process Family Engineering in Service- Oriented Applications* (PESOA) for the automotive and the electronic business (eBusiness) domain. Our work in the eBusiness domain is not an issue of this automotive focused paper. However, it is the central theme in [9], [10] and [11]. The main idea of the PESOA- approach is to expand the concept of SSF by a domain independent process model for a more detailed addressing of variable and common assets.[1]

As a result of the rapid software inclusion in automotive control units, embedded software systems are expected to account for up to 10% of the overall costs of a car [4]. Therefore, the software development and maintenance costs are now a considerable factor for an automobile, but appropriate cost models are highly insufficient [4].

Several approaches access the areas of processes, SSF or embedded systems in the automotive domain [1, 3, 6, 7, 17]. However, none of these exemplary methods measures the unadjusted size or estimates the costs of process oriented SSF in the automotive industry.

[1] Further information about the PESOA- techniques are downloadable at: www.peosa.org and www.kiebusch.de

For this reason we introduced the *Process- Family-Points* (PFP) analysis to enable a size measurement and effort estimation of embedded SSF. According to this the PFP metrics are an effort indication system to estimate the development costs of process oriented SSF in embedded control units of automobiles.

2. Real time measurement

In general real time systems differ from business information systems by the special consideration of the dimension of time [14]. An embedded real time system is always a part of a well- specified larger system, which consists of mechanical subsystems and often of a man-machine interface [13]. The control units of the automotive domain are an example application of these embedded real time systems and also the main point of the following investigations.

2.1 Real time categorization

No consistent classification of real time systems and their functional requirements is available in the actual technical literature. In many cases you will find a variant rich categorization of real time systems like in [13] and [14]. However, the following criteria enable a general economic categorization of embedded real time systems:

- Hard timing constrains are imposed and must be validated as well as guaranteed. A missed hard timing constrain is equal to a breakdown of the well-specified larger system.
- Soft timing constrains require a less rigorous validation or guarantee and are not imposed. To miss a soft timing constrain does not affect the functionality of the well- specified larger system.

We consider real time systems in SSF. Therefore, it is essential to classify the reuse of real time functions if they are a part of common or variable assets in a SSF.

Table 1 demonstrates a classification metric in order to categorize the real time functions of embedded SSF. Instead of other functional sizing methods this PFP categorization focuses the viewpoint of software developers according to the characteristics of the automotive domain.

Table 1: Classification of real time functions in embedded SSF

criterions / real time functions	asset reuse		functional validation	
	vari-able	com-mon	hard (yes)	soft (no)
variable soft (*germ.* EVW)	X	-	-	X
variable hard (*germ.* EVH)	X	-	X	-
common soft (*germ.* EGW)	-	X	-	X
common hard (*germ.* EGH)	-	X	X	-

2.2 Real time complexity weighting

Depending on the Function Point Analysis (FPA) and other sizing methods, the complexity of the categorized real time functions has to be weighted. Within this context the amount of input and output signals are an excellent complexity indicator of real time functions [12]. The metric in table 2 classifies the real time complexity according to the explained signal structure. All boundary values of this table are derived out of a *Simulink*[2] model description of a motor control unit from *DaimlerChrysler* as an example of an embedded real time SSF in [15]. This tabular metric will be calibrated for a universal usage by other SSF focused real time control units.

Table 2: Complexity matrix to rate real time functions

input / output	1 to 2	3 to 5	6 or more
1	low	average	high
2	average	average	high
3 or more	high	high	high

2.3 Real time transformation

The categorized and complexity weighted real time functions have to be converted into the temporary size measure of unadjusted PFP. This virtual size measure is comparable to unadjusted Function Points [19], the Mark II Function Point Index [20] and the Common Software Measurement International Consortium (COSMIC) functional size unit (Cfsu) [6].

As a matter of principle hard real time functions are more expensive to develop than soft real time functions because of validation and security aspects [12]. Beside this fact we have to identify if the real time functions are embedded in variable assets, which are less costly than their counterparts in common assets [12]. This is due to higher quality and security standards of common assets for the reason that they contain base functionalities which are reused in every product of the SSF. Additionally we have to separate among a horizontal perspective where

[2] Platform for multidomain simulation and model- based design of time- varying systems.

variable real time functions are encapsulated and a vertical viewpoint where one real time function can be a part of variable as well as of common assets at the same time.

To realize a high compatibility with other functional sizing methods we assign our **horizontal variable** real time categories to the complexity dependant transformation factors of the FPA data perspective {5; 7; 10; 15}. Within this activity we allocate the soft functions (EVW) to the lower and the hard functions (EVH) to the higher values as you can see in table 3.

Horizontal variable real time functions are reused relying on their product independent implementation frequency (IH, *germ. Implementierungshäufigkeit*). Hence the amount of unadjusted PFP for an EVW or an EVH is inversely proportional to their individual IH.

Historical experiences in SSF development vary between different organizations. In consequence complexity dependant correction factors for variability (KV, *germ. Korrekturfaktor- Variabilität*) are required to supplement the EVW and EVH conversion factors. If the complexity repercussions are not sufficiently covered by the conversion factors, an empirical determined KV substitutes the standard KV value. A detailed derivation of the standard value and the adjustment action as well as the updating procedure is explained in [10].

Table 3: Transforming complexity weighted, horizontal variable real time functions

complexity	unadjusted PFP	
	EVW	EVH
low *(germ.* g*)*	$\frac{KV_g \times 5}{IH}$	$\frac{KV_g \times 7}{IH}$
average *(germ.* m*)*	$\frac{KV_m \times 7}{IH}$	$\frac{KV_m \times 10}{IH}$
high *(germ.* h*)*	$\frac{KV_h \times 10}{IH}$	$\frac{KV_h \times 15}{IH}$

The determination of **horizontal common** real time functions is basically prearranged by the transformation of horizontal variable assets. Depending on the more expensive development of common assets, they are assigned to higher conversion values which are extended about two new factors. This extrapolation is executed by formula 1 which represents the original FPA- conversion factors thru a cubic function.

$$y=\frac{1}{6}\times x^3-\frac{1}{2}\times x^2+\frac{7}{3}\times x+3 \qquad (1)$$

The transformation quotients in table 4 enable the final calculation of unadjusted PFP for horizontal common real time functions with consideration of complexity (low/ average/ high), validation (soft/ hard) and historical experiences in developing common assets by a correction factor ($KG_{g/m/h}$). The reuse of common assets in every instance of a SSF is reflected by the amount of instantiated products (PA).

Table 4: Transforming complexity weighted, horizontal common real time functions

complexity	unadjusted PFP	
	EGW	EGH
low *(germ.* g*)*	$\frac{KG_g \times 10}{PA}$	$\frac{KG_g \times 15}{PA}$
average *(germ.* m*)*	$\frac{KG_m \times 15}{PA}$	$\frac{KG_m \times 23}{PA}$
high *(germ.* h*)*	$\frac{KG_h \times 23}{PA}$	$\frac{KG_h \times 35}{PA}$

As already mentioned, we have to convert also **vertical variable** real time functions. These functions are realized as a mixture of variable and common assets with a predominance of variability. Therefore, we have to determine the quota of common signals in vertical variable real time functions like it is defined in table 5.

Table 5: Common signals in a vertical variable real time function

purpose	calculation	interpretation
How high is the share of common signals in this vertical variable real time function?	$G=\frac{B}{C}$ B = Amount of common signals in this function. C = Amount of all signals in this function.	$0 \le G \le \frac{1}{2}$ Closer to ½ means more common signals in this vertical variable function.[3]

Now we have to exclude G from the variable transformation quotients in table 3 like shown in the initial part of the generic formula 2. Subsequently we have to add the product of G and the common conversion quotients from table 4 to the previous calculation according to the last part of the generic formula 2.

$$\left((1-G)\times\frac{KV_{g/m/h}\times \text{conversion factor}_{var}}{IH}\right) + \left(G\times\frac{KG_{g/m/h}\times \text{conversion factor}_{com}}{PA}\right) \qquad (2)$$

By using formula 2 to join the transformation quotients from table 3 and table 4, we are able to convert complexity weighted vertical variable real time functions into the size measure of unadjusted PFP.

The transformation of **vertical common** real time functions is prefixed by the prior procedures. Hence we have to identify the share of variable signals (V) in a

[3] The maximum value of G is ½ because a vertical variable real time function is usually characterized by a majority of variable (C-B) and a minority of common signals (B).

vertical common real time function which is realized by table 6.

Table 6: Variable signals in a vertical common real time function

purpose	calculation	interpretation
How high is the share of variable signals in this vertical common real time function?	$V = \frac{B}{C}$ B = Amount of variable signals in this function. C = Amount of all signals in this function.	$0 \le V \le \frac{1}{2}$ Closer to ½ means more variable signals in this vertical variable function.

Referring to the determination of V we have to separate this contingent of variable signals from the conversion quotients of table 4 and multiply them with the transformation quotients of table 3. The subsequent merging of these two terms is demonstrated in a general viewpoint by formula 3.

$$\left((1-V)\times\frac{KG_{g/m/h}\times \text{conversion factor}_{com}}{PA}\right) + \left(V\times\frac{KV_{g/m/h}\times \text{conversion factor}_{var}}{IH}\right) \quad (3)$$

The described categorization, complexity weighting and transformation of embedded real time functions enables an unadjusted size measurement for automotive control units. This measurement method takes account of asset reuse, validation effort, functional complexity and historical experiences from the perspective of a software developer. A practical validation of this PFP step as well as a comparison to the FPA is executed at *DaimlerChrysler Research and Technology* at the present moment. The first results are very promising since the PFP- metrics considers the reuse of common and variable assets in SSF.

3 Process measurement

A process has to be identified as a sequence of events with a definite beginning and a distinct ending. The activities within the process progression transform an object to a desired achievement [1]. The focus of this chapter is on internal software process flows instead of external development and maintenance processes which are examined in the PFP macro analysis [12].

3.1 Process categorization

Independent from the level of detail, a process is characterized by the factors of input, processing and output [16].

The PFP matrix in table 7 separates process functions depending on if they are a part of the variability or commonalities in an embedded SSF. In addition, taking in the consideration of locality, this separates them whether the input or the output factor is inside or outside the application boundary (Distinction among internal and external functionalities as well as demarcation of the software which then is measured [11]).

All processes must be logical coherent and defined from the view of the developer.

3.2 Process complexity weighting

The process complexity matrix is based on a number of generic process elements in order to support different modeling techniques. Nodes, operators and edges

Table 7: Process functions to measure the size of a SSF

criterions / process functions	asset reuse		locality			
	variable	common	input outside boundary	output outside boundary	input inside boundary	output inside boundary
variable internal process (PVI, *germ. Prozess- Variabel- Intern*)	X	-	-	-	X	X
variable unidirectional process (PVU, *germ. Prozess- Variabel- Unidirektional*)	X	-	-	X	X	-
	X	-	X	-	-	X
variable bidirectional process (PVB, *germ. Prozess- Variabel- Bidirektional*)	X	-	X	X	-	-
common internal process (PGI, *germ. Prozess- Gemeinsam- Intern*)	-	X	-	-	X	X
common unidirectional process (PGU, *germ. Prozess- Gemeinsam- Unidirektional*)	-	X	-	X	X	-
	-	X	X	-	-	X
common bidirectional process (PGB, *germ. Prozess- Gemeinsam- Bidirektional*)	-	X	X	X	-	-

characterize the universal intersecting set of conventional process models. For this reason and to ease the PFP complexity weighting, process nodes and operators are summarized together as well as analyzed jointly with their related edges.

Table 8: Complexity matrix to rate process functions

edges \ nodes	1 to 3	4 to 5	6 or more
1 to 5	low	average	high
6 to 8	average	average	high
9 or more	high	high	high

The numerical restrictions in table 8 are extracted out of a database with 43 State Machines of the Unified Modeling Language (UML) and 13 UML Activity Diagrams [15]. These processes describe an embedded software system to control a gasoline engine unit from *DaimlerChrysler*. An additional calibration of this metric is necessary and will be done during further investigations to make this table universally valid for the entire domain of embedded automotive control units.

3.3 Process transformation

A **horizontal** process describes variable **or** common functionalities of a software product in a homogeneous manner. These horizontal processes can possibly be the consequence of an optional variability capsulation inside a process orientated SSF.

The **variable** PFP transformation factors for PVI, PVU and PVB are related to the variable real time oriented PFP conversion values to achieve an internal cohesion of the PFP analysis:

- A PVI is comparatively easy to develop because of its completely internal implementation and therefore, attached to the lowest factors for a real time oriented variability conversion {5; 7; 10}.
- The realization of a PVB considers the critically requirements of variable gateways and interfaces. Consequentially the PVB conversion values are similar to the three highest real time focused transformation factors for variable assets {7; 10; 15}.
- The abstract of a PVU is a PVI- PVB- mixture and does not correspond with a single real time oriented concept. Hence the PVU conversion factors have to be created by using the linear independent, cubic formula 1 for an interpolation.

The matrix in table 9 expresses nine conversion quotients for PVI, PVU and PVB which are partly derived from formula 1. Therefore, this transformation table takes account of complexity (low/ medium/ high), locality (bidirectional/ unidirectional/ internal), asset reuse (IH) and historical experiences in SSF orientated development ($KV_{g/m/h}$).

Table 9. Transforming complexity weighted, horizontal variable process functions

complexity	unadjusted PFP		
	PVI	PVU	PVB
low *(germ*, g*)*	$\frac{KVg\times 5}{IH}$	$\frac{KVg\times 5\frac{15}{16}}{IH}$	$\frac{KVg\times 7}{IH}$
average *(germ*, m*)*	$\frac{KVm\times 7}{IH}$	$\frac{KVm\times 8\frac{5}{16}}{IH}$	$\frac{KVm\times 10}{IH}$
high *(germ*, h*)*	$\frac{KVh\times 10}{IH}$	$\frac{KVh\times 12\frac{3}{16}}{IH}$	$\frac{KVh\times 15}{IH}$

Within the high requirements of quality and modular interfaces for a generic component implementation, the horizontal **common** processes are extremely critical. For this reason the complexity dependant conversion factors for PGB, PGU and PGI are higher than their counterparts which transform the horizontal variability.

In terms of going forward, the conversion factors for horizontal common processes rely also on the real time oriented conversion factors for commonalities:

- Like the relationship between a PVI and the variable real time perspective, the PGI conversion values are identical to the lowest factors for a real time oriented transformation of common assets {10; 15; 23}.
- The consideration of common interface and gateway requirements increases the implementation size of a PGB. For this reason a PGB matches the manner of the highest real time transformation values for commonalities {15; 23; 35}.
- A single real time focused complement for the PGU concept does not exist because a PGU is a PGI- PGB-mixture. Therefore, it is necessary to interpolate additional conversion values by formula 1.

The transformation quotients in table 10 enable the final calculation of unadjusted PFP for horizontal common process functions in SSF with a consideration of the same aspects like in the transformation metric for variable processes.

Table 10. Transforming complexity weighted, horizontal common process functions

complexity	unadjusted PFP		
	PGI	PGU	PGB
low *(germ.* g*)*	$\frac{KGg\times 10}{PA}$	$\frac{KGg\times 12\frac{3}{16}}{PA}$	$\frac{KGg\times 15}{PA}$
average *(germ.* m*)*	$\frac{KGm\times 15}{PA}$	$\frac{KGm\times 18\frac{9}{16}}{PA}$	$\frac{KGm\times 23}{PA}$
high *(germ.* h*)*	$\frac{KGh\times 23}{PA}$	$\frac{KGh\times 28\frac{7}{16}}{PA}$	$\frac{KGh\times 35}{PA}$

A **vertical** process describes variable **and** common functionalities of a software product in a heterogeneous manner. To count these combined processes we separate them in variable and common assets with dependence on the strategy: *divide et impera* - divide and conquer.

Vertical **variable** processes contain a preponderance of variable functions. Consequently the table 9 for calculating horizontal variability is also the main foundation to count vertical variable processes.

According to this heterogeneity of vertical variable assets it is necessary to determine the share of common elements (G) in this vertical variable processes like it is illustrated in table 11.

Table 11: Calculating the proportion of common elements in a vertical variable process

purpose	calculation	interpretation
How high is the share of common nodes and edges in this vertical variable process?	$G=\frac{B}{C}$ B = Amount of common elements in this process. C = Amount of all elements in this process.	$0 \leq G \leq \frac{1}{2}$ Closer to ½ means more common elements in this vertical variable process.

Formula 2 enables beside the discussed real time conversion also the final transformation of vertical variable processes in embedded control units. The common nodes and edges which are represented via G have to be removed from the variable oriented calculations of table 9 like in the first fraction of the mentioned equation. Afterwards these excluded elements must be counted like horizontal commonalities based on table 10. This is accomplished by the second part of formula 2.

As the opposite of vertical variability, vertical **common** processes include a majority of common functions. Table 10 is a metric to count horizontal commonalities and also the main component to measure vertical common processes.

At the beginning of this transformation we have to clarify the share of variable nodes and edges (V) in vertical common processes according to the model in table 12.

Table 12: Calculating the proportion of variable elements in a vertical common process

purpose	calculation	interpretation
How high is the share of variable nodes and edges in this vertical common process?	$V=\frac{B}{C}$ B = Amount of variable elements in this process. C = Amount of all elements in this process.	$0 \leq V \leq \frac{1}{2}$ Closer to ½ means more variable elements in this vertical common process.

The reuse of formula 3 allows the concluding conversion of vertical common processes in embedded software systems. After the identification of variable elements through V, it is essential to remove these nodes and edges from the calculations of table 10 like it is shown in the initial section of formula 3. For a complete representation of a vertical common process it is indispensable to add the variable elements by using the transformations quotients from table 9 as it is displayed in the last fraction of the generic formula 3.

3 Exemplary case study

To complete and summarize the previous theoretical investigations we apply these PFP metrics to a restricted practical case study. Figure 1 represents an embedded real time function to administer the air condition by an engine control unit. An embedded process to regulate the throttle valve is additionally illustrated in figure 2.

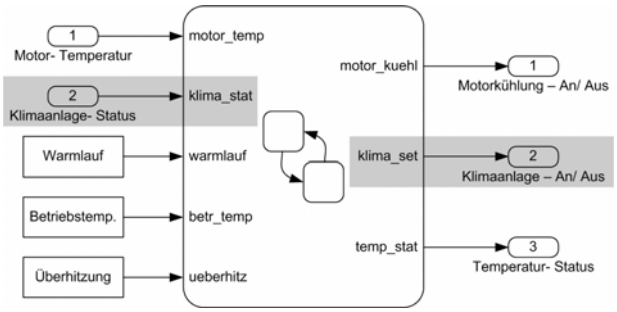


Figure 1: Exemplary real time function from an embedded SSF

The **real time function** from figure 1 has to be transformed into the temporary size measure of unadjusted PFP. Beside this diagram we need the following basic information to use the discussed PFP micro analysis in this measurement situation:

- Soft timing constrains;
- Vertical variability with two variable (highlighted gray) and six common signals;
- High complexity in dependence on the signal structure and table 2;

- Quota of variable elements is V = $^2/_8$ = 0,25 according to table 6;
- Standardized correction factors ($KG_{g/m/h}$ = 1; $KV_{g/m/h}$ = 1) are used because of unavailable historical experiences;
- Variable elements are present in two instances of a SSF including tree products.

The size of this embedded vertical common, soft real time function accounts seven unadjusted PFP according to formula 3.[4]

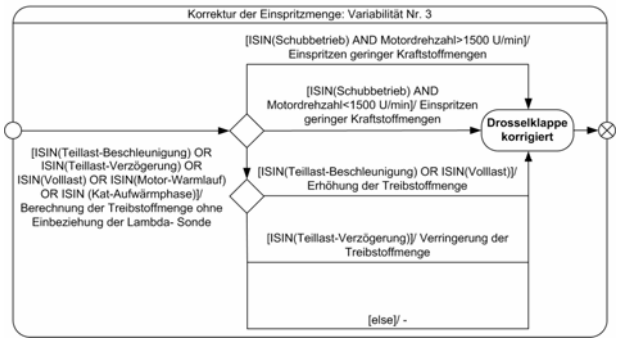


Figure 2: Exemplary process function from an embedded SSF

For a subsequent conversion of the **process function** in figure 2 we need the following aspects to apply the explained PFP micro analysis:

- External input "revolution" (*germ. Motordrehzahl)* and external output "throttle position" (*germ. Drosselklappenöffnung*);
- Horizontal variability modeled in an encapsulated variable asset;
- Average complexity according to the amount of nodes/ edges and table 3;
- Standardized correction factors ($KV_{g/m/h}$ = 1) are reused because of unavailable historical experiences;
- Implementation of the variable process in one instance of the SSF.

The size of this embedded horizontal variable, bidirectional process embraces ten unadjusted PFP in dependence on table 9.[5]

The PFP sums of all measured real time functions and processes in an embedded automotive control unit describe the unadjusted size of a previously defined counting scope[6] [11]. For a counting scope in the size of the brief example above we would count 17 unadjusted PFP.

The sum of all unadjusted PFP describes a size measure of an embedded control unit and can be used as a coarse effort indicator for a SSF development [8]. Finally we have a size measurement system and therefore a base to build an effort estimation tool for embedded SSF in the automotive sector.

4 Conclusion and further research

The metrics in this article are independent from the development techniques of process orientated SSF and accomplish an unadjusted size measurement for embedded control units in the automotive domain.

To establish an ISO/ IEC 14143 compatible measurement system was not the goal of our PFP approach. This is justified in that we need to analyze the viewpoint of a software developer instead of the consumer perspective in the automotive sector. Furthermore we appreciate the flexibility to extend our approach with a PFP macro analysis which is not compliant to the generic rules of functional sizing in ISO/ IEC 14143 but helpful for accurate cost estimation.

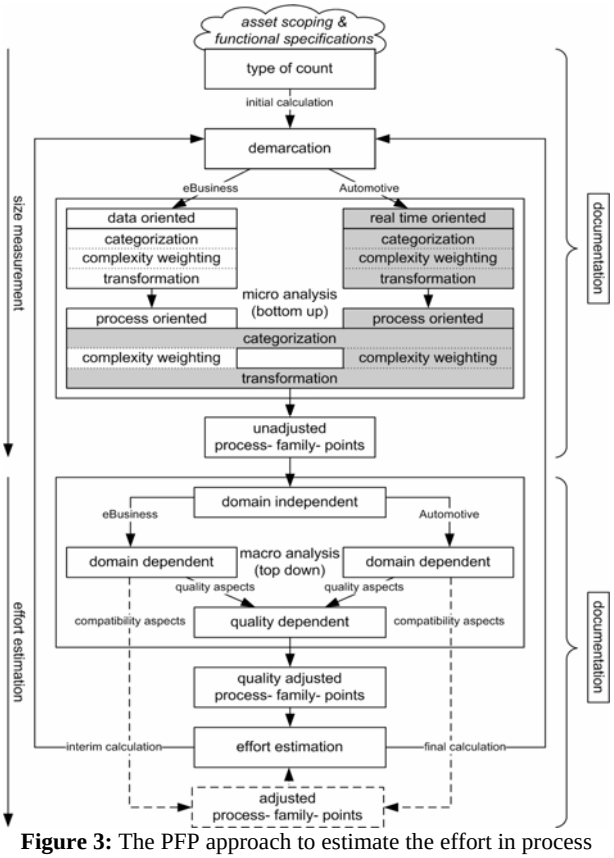


Figure 3: The PFP approach to estimate the effort in process oriented SSF

4 $\text{unadjusted PFP} = \left((1-V)\times\frac{KG_h\times 23}{PA}\right) + \left(V\times\frac{KV_h\times 10}{IH}\right)$

$= (1-0,25)\times\frac{1\times 23}{3} + 0,25\times\frac{1\times 10}{2} = 7$

5 $\text{unadjusted PFP} = \frac{KV_m\times 10}{IH} = \frac{1\times 10}{1} = 10$

6 "Application independent border which can be embrace more or less functionality as a single software program" [11].

Figure 3 illustrates the entire PFP concept to estimate the effort for process oriented SSF in multiple domains. The reason that the PFP analysis focuses the eBusiness as well as the automotive sector is that both domains are characterized by processes. Therefore, a process orientation increases the efficiency in a domain specific SSF development and reduces the costs of software products in these exemplary sectors.

Within the first PFP step we identify the type of count which can be a development or a reuse project as well as a count for a single product of the SSF [9]. The following PFP action which is illustrated in figure 3 defines the application boundary and the counting scope to demarcate the process oriented SSF [11].

An unadjusted size measure is calculated by the eBusiness oriented PFP micro analysis in figure 3. Within this framework we categorize, weight and transform a data as well as a process oriented view. In [9], [10] and [11] detailed description of these PFP parts may be found.

The real time and process oriented PFP modules (categorization, complexity weighting and transformation) with a grey emphasis are elucidated in this article and determine an unadjusted size measure for embedded SSF in the automotive sector.

A completion of the domain specific PFP micro analysis is to be gained by the PFP macro analysis which considers external influences on development costs with soft characteristics. The PFP macro analysis allows a flexible modification and extension of all influencing factors. Therefore, this part of the PFP analysis is versatile applicable and could be used as a potentional substitution of the FPA or Mark II adjustment procedures.

Inside of the PFP macro analysis we examine twenty domain independent influences according to the approaches of FPA, Mark II and in allusion to the Object/ Data Point Method. Subsequently we look at fifteen domain specific system characteristic for the eBusiness as well as for the automotive domain. After these obligatory steps it is possible to calculate adjusted PFP as a size measure which is comparable to other adjusted functional size measures. Alternatively we can determine quality adjusted PFP by regarding quality aspects in dependence on ISO/ IEC 9126. With this action we loose our compatibility to FPA and Mark II but realize a more precise measure for an exact effort estimation in process oriented SSF.

The procedure of estimating SSF development costs is executed before (initial calculation), in between (interim calculation) and after (final calculation) the software engineering project according to figure 3. This enables the consideration of SSF evolution between variable and common assets. Furthermore it is possible to detect additional functionality that was not specified in the requirements but identified during development.

The additional documentation process is an integral part of the PFP approach in order to retrace prior calculations and to support future effort estimations.

To validate the PFP effort estimation system we developed a PFP counting tool and apply our approach currently at a SSF oriented project with *DaimlerChrysler Research and Technology* and in an eBusiness project seminar at the *University of Leipzig*. The first results in terms of a comparison between the PFP analysis and other sizing methods such as the FPA or COSMIC FFP are very promising. The PFP approach covers the characteristics of SSF much better than the FP or COSMIC FFP analysis particularly with regard to the special aspects of reuse, processes and general system characteristics.

During the validation process we will derive regression lines for both domains in order to forecast the effort of developing process oriented SSF. With the associated two dimensional ([quality-] adjusted PFP; development effort) diagram we are able to predict software development costs after the very early project phase of asset scoping.

References

[1] Abran, A. *Functional Size Measurement for Real Time and Embedded Software.* In: Fourth International Symposium and Forum on Software Engineering Standards, Curitiba May 1999.

[2] Allgaier, H. J., *Segmentierung der Auftragsabwicklung: Modellanalyse einer Gestaltungskonzeption*, PhD- Thesis, Technische Universität München, München 1994.

[3] Beuche, D. *Composition and Construction of Embedded Software Families.* Dissertation, Otto-von-Guericke-Universität, Magdeburg 2003.

[4] Broy, M. *Automotive Software Engineering.* In: Proceedings of the 25th International Conference on Software Engineering (ICSE'03), Portland, May 2003.

[5] Clements, P., Northrop, L. *Software Product Lines: Practices and Patterns.* Addison- Wesley, Boston et al. 2002.

[6] Commom Software Measurement International Consortium (Ed.), *COSMIC- FFP: Measurement Manual 2.2*. Québec 2003.

[7] Diaz- Herrera, J., L., Madisetti, V., K. *Embedded Systems Product Lines.* In Proceedings of Software product lines, CSE Workshop. Limerick, June 2000.

[8] Kamm, C., Siedersleben, J., Schick, D., Saad, A. *Systematische Aufwandsschätzung für Software im Fahrzeug.* In: OBJEKTspektrum Nr. 6, November/ Dezember 2004, pp. 60-64.

[9] Kiebusch, S. *An approach to a data oriented size measurement in Software- Product- Families.* In: Abran, A., Bundschuh, M., Dumke, R., Ebert, C., Zuse, H. (Hrsg.)

Metric News: Journal of the GI- Interest Group on Software Metrics, Vol. 9, Nr. 1, August 2004, pp. 60-67.

[10] Kiebusch, S. Franczyk, B. *Functional size measurement of processes in Software- Product- Families.* In: Proceedings of the 2nd Software Measurement European Forum, Rome, March 2005, pp. 161-172.

[11] Kiebusch, S. *Towards a Function-Point oriented analysis of process focused Software-Product-Families.* In: Proceedings of the Net.ObjectDays 2004: 5th Annual International Conference on Object- Oriented and Internet-based Technologies, Concepts, and Applications for a Networked World, Erfurt, September 2004, pp. 147-152.

[12] Kiebusch, S., Richter, E., Weiland, J. *Metriken: Definition und Validierung.* PESOA TR x/ 2005, Universität Leipzig, To be published at: http://www.pesoa.org, Leipzig 2005.

[13] Kopetz, H. Real- Time Systems: *Design Principles for Distributed Embedded Applications.* Kluwer Academic Publishers, Boston et al. 1997.

[14] Lauber, R., Göhner, P. *Prozessautomatisierung 1: Automatisierungssysteme und -strukturen, Computer- und Bussysteme für die Anlagen- und Produktautomatisierung, Echtzeitprogrammierung und Echtzeitbetriebssysteme, Zuverlässigkeits- und Sicherheitstechnik.* 3. Edition, Springer- Verlag, Berlin et al. 1999.

[15] Richter, E., Schnieders, A., Weiland, J. *Prozessanalyse und –modellierung in der Domäne Automotive*. PESOA TR 07/ 2005, DaimlerChrysler Research and Technology and Hasso- Plattner- Institut, http://www.pesoa.org, Ulm and Potsdam 2005.

[16] Scholz, R., and Vrohlings, A. *Prozeß- Struktur-Transparenz.* In: Gaintanides, M., Scholz, R., Vrohlings, A., and Raster, M. (Ed.), Prozeßmanagement: Konzepte, Umsetzungen und Erfahrungen des Reengineerings, München 1994, pp. 37-56.

[17] Sholom, C., Zubrow, D., Dunn, E. *Case Study: A Measurement Program for Product Lines.* Technical Note, CMU/ SEI-2004-TN-023, Software Engineering Institute, Carnegie Mellon University, Pittsburgh 2004.

[18] Speck, A. Reusable Industrial Control Systems. *Industrial Informatics*. Special Section of the IEEE Transactions on Industrial Electronics Society, 50(3), pp 412-418, 2003.

[19] The International Function Point Users Group (Ed.) *Function Point Counting Practices Manual: Release 4.2.* Clarkston 2004.

[20] United Kingdom Software Metrics Association (Ed.) *MK II Function Point Analysis: Counting Practices Manual Version 1.3.1.* Kent 1998.

Evaluating the Impact of a New Technology Using Simulation: The Case for Mining Software Repositories

David Raffo, *Member IEEE*, Tim Menzies, *Member IEEE*

***Abstract*— Adopting new technologies on a development process can be a risky endeavor. Will the project accept the new technology? What will be the impact? Far too often the project is asked to adopt the new technology without planning how it will be applied on the project or evaluating the technology's potential impact. In this paper we provide a case study evaluating one new technology. Specifically we assess the merits of defect detectors learned from software repositories of static code measures (Halstead and McCabe). Using process simulation, we find situations where the use of such detectors is *useful* and situations where the use of such detectors is *useless* for large-scale NASA projects that utilize a process similar to the IEEE 12207 systems development lifecycle.**

***Index Terms*—Software Process Simulation, Technology Adoption, Technology Evaluation, Data Mining**

I. INTRODUCTION

Good research results are wasted unless there is a compelling business case to use them. Without such a case, a project manager may not be convinced that they should, for instance, reallocate scarce resources to implement a new technology on their project. The aim of this paper is to offer one example of a business case evaluating the impact of a new technology currently under development using software process simulation.

Process simulation was selected as a means to evaluate this new technology because it captures the details associated with how the new technology would actually be applied in the field. Moreover, process simulation allows for extensive sensitivity analysis which enables different scenarios and special case conditions to be checked. This kind of analysis gave the developers of the technology realistic feedback about their targeted field of use and their envisioned approach for roll out of the tool.

The new technology selected for this evaluation was defect detectors learned from software repositories of static code measures (Halstead and McCabe**)**. As will be discussed in the background section, recent research advances by Menzies et al. [1] have significantly improved the probability of detection of defects, lowered the probability of false alarms while at the same time reducing the amount of code that must be manually inspected in order to achieve these results. Given these improvements, it seems possible that this new technology would be "ready for prime time". Software Process Simulation (SPS) was used to assess whether this was in fact the case. More importantly, SPS was used to identify under what conditions the technology was "ready for prime time" and under what conditions it was not.

Although a variety of methodologies where used to conduct the evaluation of this new technology including the Process Tradeoff Analysis Method (PTAM) [2], general simulation methods regarding sensitivity analysis and portions of the Broad Range Sensitivity Analysis approach (BRSA) [3]. We believe the main contribution of this paper is a practical one dealing with technology planning, application and adoption. We see that this case study provides strong evidence of the benefit of applying process simulation in industrial settings.

As mentioned, section 2 (background) will provide an overview of the process simulation model and approach used in this case study as well as recent research advances by Menzies et al. In section 3, we provide an overview of the case study performed. Due to space constraints, we have only presented two scenarios in this paper – the first scenario showing an improvement in overall project performance and the section scenario showing and overall reduction in performance. In section 4, we present case study results for the two scenarios selected. Finally, we conclude in section 5.

II. BACKGROUND

A. Process Simulation and the Model Used for this Study

Process simulation is commonly used in many industries including manufacturing and service operations to address these kinds of issues. In

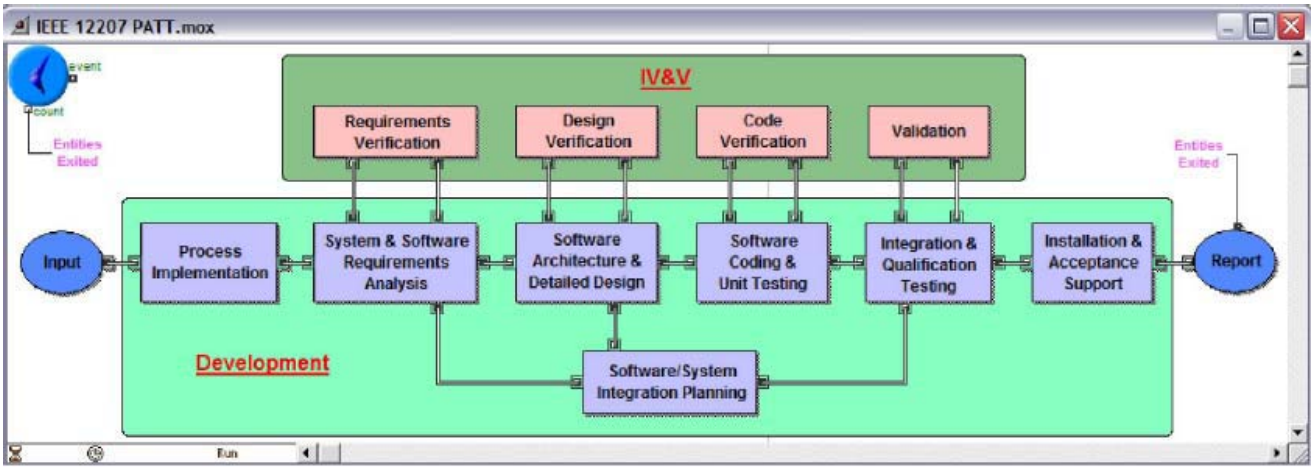


Figure 1 – IEEE 12207 Process Simulation Model with IV&V Layer

recent years, process simulation has been applied to software development processes[1]. The key advantage to process simulation is that these models can capture the details associated with the development process and provide a systematic approach for incorporating metrics data and creating the necessary process level predictions along multiple measures of performance [4], [5], [6]. Simulation modeling tools (e.g. Arena, Extend, Stella, etc.) simplify conducting sensitivity (or "what if") analyses. As a result, process simulation models can explicitly capture localized changes to the process made by implementing a new tool, technology or method and then predict the overall project-level impacts.

In this study, we employ process simulation to assess the impact of applying learned fault detectors under multiple possible operational scenarios. The specific process simulation model used for this study is a model of the IEEE 12207 systems development process [7]. This process is representative of the process used on large-scale NASA and US Department of Defense (DoD) projects. The model contains industry standard benchmark data from [8] for largescale systems development. Moreover, the model has been tuned using a data set from 8 NASA projects over 100 KSLOC in size. Predictions made with the model provide similar accuracy to those obtained using COCOMO I (i.e. predictions were within 30% of actual values, more than 70% of the time).

Figure 1 shows a top-level view of the software development model used for this study. As can be seen in that figure, the main life cycle phases of the IEEE 12207 process are:

- Process implementation
- System and software requirements analysis
- Software architecture and, detailed design
- Software coding and unit testing
- Software and system integration planning
- Integration and qualification testing
- Integration and acceptance support

Figure 1 also shows that we have augmented IEEE 12207 with an additional IV&V layer that models the actions of external consultants auditing software artifacts.

B. The New Technology: Learned Defect Detectors

Particular measures can be assessed by combining them to form detectors and then assessing each *specific* detector using: (1) PD: the probability of defecting faults (a.k.a. recall), (2) Accuracy, (3) The effort associated with the detector triggering, and (4) The probability of false alarm (PF). Ideally, a detector has a high probability of detection and a low false alarm rate but, in practice, increasing one also increases the other.

The art of data mining is finding some way to trade-off competing PDs and PFs. Numerous data mining algorithms have been developed that

[1] See the proceedings of the *ProSim International Workshops*, at `http://www.prosim.pdx.edu/` and special issues of the *Journal of Systems and Software* (Vol 46, No 2/3, Vol. 47, No. 9 and Vol, 59, No. 3), and the international journal of *Software Process: Improvement and Practice*, Vol 5, No. 2/3, Vol. 7, No. 3/4 and Vol 9, No. 2) on this topic.

achieve this goal. Elsewhere Menzies et,al. have compared several widely-used methods [1] and concluded that Bayesian methods usually out-perform decision-tree methods, at least for the data sets studied here.

A *stratification effect* was also seen in those studies; i.e. learning on specialized subsets (e.g. just examples from a particular sub-sub-system) is often preferred to learning from the entire system.

Those studies also saw a previously unreported effect. It is well known that the efficacy of the learned theories increases as the number of training examples increases. However, surprisingly, that increase tapers off remarkably early for the NASA defect data. In fact, given a binary classification problem (ie. "true" means "defects > 0" and "false" means "defects=0"), no statistically significant increase was seen after training *from only 50 examples.*

Taken together, these results suggest that it would be relatively simple and quick to learn specialized detectors for local domains from data collected just from, say, a sub-sub-system. The rest of this paper uses process simulation to see the effects of this policy.

Before turning to the business case simulations, we need to state some underlying assumptions. One important issue is the meaning of PD (probability of detection). Our research results show PDs learned from defects logs containing issue reports from multiple sources: inspections, software tests, hardware tests, formal method results, etc. As a result of this extensive checking, we assert that the PDs shown above are equal to the percentage of defects *ever* found in the system. Another kind of PD, would be that seen in data miners executing on just the issue reports seen in the most recent manual inspections. In this case, the PD changes to a smaller value (which we denote PD0) since our learners might only find a certain percentage of the defects contained in the inspection logs which, in turn, is some percentage of the total number of defects.

Further research is required to determine the *actual* value of the PD0 learned from just inspection logs. While we await that data, in this paper, we present two scenarios. In Scenario I, we assume PD refers to a percentage of the total number of errors; i.e. the "IV&V situation" where we are learning from rich defect logs. In Scenario II, we will assume PD0 i.e. the "V&V situation" where we are only learning from the results of inspections.

Scenario I's conclusions will endorse using defect detectors while Scenario II's conclusions will be more negative. We also make several other assumptions, based on the results of [1, 9 and 10]:

A1: Based upon the early plateau effect [1], we will use 50 as the number modules needed for data miners to learn defect detectors at the sub-sub-system level.
A2: Effort associated with our detectors being triggered is linearly proportional to line of code; i.e. Effort = PD + 5% . . . 10%.
A3: For our detectors, PF ≤ 10%.
A4: PD associated with the detectors; i.e. PD = 40. . .50%.
A5: Assumption A4 assumes, in turn, that defect detectors are learned from data divided below the sub-system level.

In principle, these assumptions can be checked and adjusted as necessary. This is one of the strengths of process simulation. Our current process model has been built and repeatedly checked over the last three years.

III. BUSINESS IMPLICATIONS OF DEFECT DETECTORS

A. Baseline Model Results

Baseline performance was predicted in terms of development effort (or cost), effort devoted to rework, IV&V effort, project duration, corrected defects, and escaped (or delivered) defects. Our baseline of the AS-IS process presumes:

1. The project is 100,000 lines of code.
2. Figure 1 shows the assumed software process: i.e. IEEE 12207+IV&V model.
3. Full Fagan inspections [11] are done at all development phases- including at the coding phase. These manual inspections find 40% to 60% of the total defects.

The actual baseline performance for the AS-IS process (without using data miners applied to defect detectors as part of IV&V activities) can be seen in Figure 2.

B. Scenario I: Defect Detectors and IV&V

For the first TO-BE process scenario, we apply the defect detectors as part of an independent verification and validation (IV&V) step after coding and code inspections are complete. In this TO-BE scenario, defect detectors are utilized in IV&V work as follows:

- Defect logs and code modules that have completed code inspections and other forms of testing are sent to IV&V.

	Total Size (KLOC)	Total + Effort IV&V (PM)	Total Effort (PM)	Total Rework Effort (PM)	Total Dura-tion (Month)	Average Du-ration	Total Defect Corrected	Total Latent Defects	Code Inspection Effort (PM)
Baseline									
Average	99.79	781.41	781.41	160.56	32.81	28.51	5,907.08	507.81	9.67
Std Dev	4.00	27.66	27.66	6.95	1.43	1.22	257.00	22.00	0.29
Case 1: detcap IV&V = 0									
Average	99.79	782.57	781.41	160.56	32.85	28.51	5,907.08	507.81	10.83
Std Dev	4.00	27.67	27.66	6.95	1.43	1.22	257.00	22.00	0.30
Deltas		-1.16 ◄	0.00	0.00	-0.05	0.00	0.00	0.00 ◄	-1.16
Case 2: detcap IV&V = 0.01									
Average	99.79	782.06	780.90	160.05	32.85	28.50	5,928.55	505.79	10.83
Std Dev	4.00	27.67	27.65	6.93	1.43	1.22	258.00	22.00	0.30
Deltas		-0.65 ◄	0.51	0.51	-0.04	0.00	-21.47	2.02 ◄	-1.16
Case 3: detcap IV&V = 0.02									
Average	99.79	781.27	780.11	159.29	32.83	28.49	5,931.95	502.40	10.83
Std Dev	4.00	27.66	27.64	6.89	1.43	1.22	258.00	21.00	0.30
Deltas		0.14 ◄	1.30	1.27	-0.03	0.02	-24.87	5.41 ◄	-1.16
Case 4: detcap IV&V = 0.03									
Average	99.79	780.48	779.32	158.53	32.82	28.47	5,935.34	499.00	10.83
Std Dev	4.00	27.64	27.63	6.86	1.43	1.22	258.00	21.00	0.30
Deltas		0.93 ◄	2.09	2.03	-0.01	0.03	-28.26	8.81 ◄	-1.16
Case 5: detcap IV&V = 0.04									
Average	99.79	779.68	778.52	157.77	32.80	28.46	5,938.73	495.61	10.83
Std Dev	4.00	27.64	27.62	6.83	1.43	1.22	258.00	21.00	0.30
Deltas		1.72 ◄	2.88	2.79	0.00	0.05	-31.65	12.20 ◄	-1.16
Case 6: detcap IV&V = 0.05									
Average	99.79	778.89	777.73	157.01	32.79	28.44	5,942.13	492.22	10.83
Std Dev	4.00	27.63	27.61	6.80	1.43	1.21	259.00	21.00	0.30
Deltas		2.51 ◄	3.67	3.55	0.02	0.06	-35.05	15.59 ◄	-1.16
Case 7: detcap IV&V = 0.10									
Average	99.79	774.94	773.78	153.21	32.72	28.37	5,959.10	475.24	10.83
Std Dev	4.00	27.58	27.57	6.63	1.43	1.21	259.00	20.00	0.30
Deltas		6.47 ◄	7.63	7.35	0.09	0.14	-52.02	32.57 ◄	-1.16

Fig 2. Operational Scenario I: Using Defect Detectors in IV&V Mode

- Defect detectors are learned on the logs and then applied to 100% of the code. Once the logs are in a format suitable for the learners, this can be done automatically and quickly (a mere matter of seconds). In our simulations we make the conservative assumption that preparing the input logs takes two person days(or 16 hours of effort) for a large 100 KSLOC project.
- The defect detectors identify code modules that are likely to contain defects. Since the code modules will have gone through code inspections and other assessment measures during project level V&V, many of the modules that are identified will already be known to contain defects. These modules will not be looked at again. Instead, the defect detectors will be used to identify modules where no defects were found during their initial code inspections, but whose characteristics indicate that these modules are likely to have defects.
- The modules that trigger the detectors are then reinspected. The re-inspection rate's upper bound is PF; i.e. 10% (from [A5]).

To assess the impact of learning data miners for defect detectors, then using them in IV&V mode, the next parameter required is an estimate of the percentage of the escaped defects that will be found using the above procedure. At present, more research is necessary to empirically determine this percentage. While we await those results, we can use the process simulation model to identify the minimum percentages required in order to break even (where expenses equal benefits). Moreover, the process simulation model can help assess the risk of applying the defect detectors by examining the worst-case scenario (i.e. when no additional defects are detected).

The results of these tests are shown in Figure 2 and, for the purposes of this discussion, we focus on the cells marked with a black triangle (◄). These cells show the difference between the baseline data and the results from Scenario I. As

	Total Size (KLOC)	Total Effort (PM)	Total Rework Effort (PM)	Total Duration (Month)	Average Duration	Total Defect Corrected	Total Latent Defects	Code Inspection Effort (PM)
Baseline								
Average	99.79	781.41	160.56	32.81	28.51	5,907.08	507.81	9.67
Std Dev	4.00	27.66	6.95	1.43	1.22	257.00	22.00	0.29
Case 2: detcap IV&V = 0.48								
Average	99.79	780.24	163.24	32.74	28.44	5,895.33	519.56	5.52
Std Dev	4.00	27.52	7.07	1.43	1.21	257.00	22.00	0.14
Deltas		1.17 ◄	-2.68	0.07	0.07	11.75	-11.75 ◄	4.15
Case 1: detcap IV&V = 0.47								
Average	99.79	781.53	164.58	32.78	28.47	5,889.45	525.43	5.42
Std Dev	4.00	27.54	7.13	1.43	1.22	256.00	22.00	0.14
Deltas		-0.13 ◄	-4.02	0.03	0.03	17.63	-17.62 ◄	4.24

Fig 3. Operational Scenario II: Using Defect Detectors in V&V Mode.

can be seen, using defect detectors breaks-even (i.e. the Delta goes from negative to positive) if the above approach can detect an additional 1 to 2% of the latent defects in the code and starts showing a positive benefit in both effort and latent defects at 3%. The worst case is that an additional 1.16 person months would be expended doing inspections that do not find any new defects.

Moreover, if 5% or 10% of the latent defects are found, the quality of the code would be improved by an average 15.5 and 32.5 defects respectively and an average 2.5 and 6.5 person months of effort respectively could be saved. To repeat, the *minimum* performance target for defect detectors to be beneficial for IV&V would be 3% additional defects detected and the *maximum* exposure would be 1.16 person months of effort. Based on our commercial work with companies exploring re-inspections of their code, we are confident that the 3% additional defects threshold can be easily surpassed. However, in this forum, we cannot publish supportive evidence for this claim since it is based on proprietary data.

C. Scenario II: Defect Detectors and Inspection-based V&V

Scenario II is the case where analysts have a *much weaker* training set; i.e. *only* the results of internal inspections. This is the PD0 case where the data miners find only *some* the defects in defect logs which contain only *some* of the project defects. One situation where this could happen would be when a team declines to wait for the IV&V team to report issues. Instead, they use their own experience of, say, their code inspections to build defect detectors. For example:

- A minimum set of code was inspected. From assumption A3, 50 modules per sub-sub-system, or 11.5% of the code, would be required to achieve plateau performance.
- Using the defect logs from these inspections and the inspected modules, defect detectors were learned using data miners and applied to the rest of the code.
- Modules were identified as likely candidates for defects.
- Only those portions of the code that are tagged as likely ”hot spots” were inspected.

Under this scenario, the defect detectors would select 61.5% of the code for further inspection. This would result in 38.5% reduction in inspection effort (approximately 3.7 person-months) and inspection schedule savings. However, process simulation shows that the savings in inspection effort would not offset the increase in defect detection and rework costs associated with finding these defects later in the development process. Figure 3 shows the baseline results and results of having an expected defect detection capability of 47% which will cause the process using defect detectors during project V&V to break even on effort (but have an overall poorer quality).

If we are learning on an inspection log containing 50% of all defects (the expected case), then such a 47% overall defect detection rate is only possible if a data miner can learn near-perfect detectors with a PD of 98% (i.e. 50% * 98% = 47%). Since this is highly unlikely (to say the least), the conclusion of Scenario II must be to doubt the value of defect detectors for improving V&V.

IV. Conclusion

Process simulation is a powerful tool for conducting what-if queries on software processes. We have shown above two such what-ifs: in Scenario I we did not know the impact of applying data miners to IV&V yet, in Figure 7, we could still identify the break even point where such mining was useful.

More generally, we see this kind of analysis as being very beneficial for more than just assessing defect detectors learned from software repositories. The above process simulation is an example of a general technological assessment process where we can:

- Identify the conditions under which application of a new technology would be beneficial.
- As importantly, we can identify situations when applying this technology would *not* be beneficial.
- We can have performance benchmarks or criteria that vendors of a new technology would need to achieve in order for an organization to consider investing and adopting their technology.
- We can diagnose problems associated with implementing a new tool or technology and identify new and creative ways to apply the technology to the benefit of the organization (and the vendors)
- Finally, we can do all this *before* the technology is purchased or applied and therefore can save scarce resources available for process improvement.

Acknowledgment

This research was conducted at West Virginia University, Portland State University, partially sponsored by the NASA Office of Safety and Mission Assurance under the Software Assurance Research Program led by the NASA IV&V Facility. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government.

References

[1] T. Menzies and J. DiStefano and A. Orrego and R. Chapman, “Assessing Predictors of Software Defects", Proceedings, workshop on Predictive Software Models, Chicago", 2004, Available from http://menzies.us/pdf/04psm.pdf

[2] D.M. Raffo, “Modeling software processes quantitatively and assessing the impact of potential process changes of process performance,” May 1996, Ph.D. thesis, Manufacturing and Operations Systems.

[3] Wakeland, Martin, and Raffo, “Using Design of Experiments, Sensitivity Analysis, and Hybrid Simulation to Evaluate Changes to a Software Development Process: A Case Study”, *Software Process: Improvement and Practice*, Vol 9, No 2, 2004

[4] [7] D. Raffo and M. Kellner, “Impact of potential process changes: A quantitative approach to process modeling,” in Elements of Software Process Assessment and Improvement, K. El Emam and N. Madhavji, Eds. 199, IEEE Computer Society.

[5] G.A. Hansen, Automating Business Process Reengineering, Prentice Hall, 1997.

[6] M. Laguna and J. Marklund, Business Process Modeling, Simulation, and Design, Pearson Prenctice Hall, 2004.

[7] Institute of Electrical and Inc. Electronics Engineers, “Iso/iec 12207 standard for information technology - software lifecycle process,” 1998.

[8] C. Jones, Applied Software Measurement (second edition), McGraw Hill, 1991.

[9] T. Menzies, J. Di Stefano, K. Ammar, K. McGill, P. Callis, R. Chapman, and Davis J, “When can we test less?,” in IEEE Metrics’03, 2003, Available from http://menzies.us/pdf/03metrics.pdf.

[10] Tim Menzies and Justin S. Di Stefano, “How good is your blind spot sampling policy?,” in 2004 IEEE Conference on High Assurance Software Engineering, 2003, Available from http://menzies.us/pdf/03blind.pdf.

[11] M. Fagan, “Design and code inspections to reduce errors in program development,” IBM Systems Journal, vol. 15, no. 3, 1976.

A Software Process Simulation Model of Extreme Programming

Marco Melis, *Student Member, IEEE* Ivana Turnu, *Student Member, IEEE* Alessandra Cau and Giulio Concas

***Abstract*—In this paper we present a simulation model that we have developed to evaluate the applicability and effectiveness of Extreme Programming (XP) process. The XP process has been modelled and a simulation executive has been written, enabling to simulate XP software development activities. The model follows an object-oriented approach, and has been implemented in Smalltalk language, following an XP process itself. It will be able to vary the usage level of some XP practices and to simulate how all the project entities evolve consequently. Here we present some results concerning the simulation of an XP process varying the adoption level of the Pair Programming practice.**

I. Introduction

EXTREME Programming (XP) is a new lightweight methodology of software development which has become very popular in recent years, and which has been recently reviewed [1]. The effectiveness of XP practices, however has still to be quantitatively assessed. Works that report quantitative results are still very scarce and propose empirical studies which are extremely costly and virtually impossible to be performed at a large degree of completeness. Therefore, the use of simulation to estimate and verify the effectiveness of a particular development process represents an alternative solution.

In this paper we present a simulation model we have developed in order to evaluate the applicability and effectiveness of XP process.

A simulation executive has been written, enabling to simulate XP software development activities. The model follows an object-oriented approach, and has been implemented in Smalltalk language, following XP process itself. It will be able to vary the usage level of some original XP practices, which are still present also in the second edition of Beck's book [1], such as *Real Customer Involvement* (On-site customer), *Pair Programming*, *Test-First Programming*, *Continuous Integration*, and to simulate how all the project entities evolve consequently.

Presently, the modules simulating the planning and code production activities are fully operative while others are under development. We are currently able to simulate the influence of Pair Programming and Test Driven Development practices. Here we present some results concerning the simulation of these modules.

The remainder of the paper is organized as follows: in section II we give some information on the Software Process Simulation approach; in section III an overview of the Extreme Programming and of the Pair Programming practice in particular is reported. Section IV is aimed to expose some important related works on the simulation modeling of XP, while in sections V and VI we describe our model. Sections VII and VIII report data and results obtained from our simulator.

Department of Electrical and Electronic Engineering
Università di Cagliari, Cagliari, 09123, Italy
Phone: (+39) 070-6755774, email: marco.melis@diee.unica.it
Department of Electrical and Electronic Engineering
Università di Cagliari, Cagliari, 09123, Italy
Phone: (+39) 070-6755774, email: ivana.turnu@diee.unica.it
Department of Electrical and Electronic Engineering
Università di Cagliari, Cagliari, 09123, Italy
Phone: (+39) 070-6755774, email: alessandra.cau@diee.unica.it
Department of Electrical and Electronic Engineering
Università di Cagliari, Cagliari, 09123, Italy
Phone: (+39) 070-6755781, email: concas@diee.unica.it

II. Software Process Simulation

Software Process Simulation (SPS) is becoming increasingly popular in the software engineering community, both among academics and practitioners [2]. In fact, new and innovative software engineering techniques are being developed constantly, so a better understanding of them is useful to evaluate their effectiveness and to predict possible problems. Simulation can provide information about these issues avoiding real world experimentation, which is too costly in terms of time and money. This field of SPS has attracted growing interest over the last twenty years, but only in recent years it is beginning to be used to address several issues concerning the strategic management of software development and process improvement support. It can also help project managers and process engineers to plan changes in the development process. The development of a simulation model is an inexpensive way to collect information when costs, risks and complexity of the real system are very high.

In order to create a connection between real world and simulation results, it is usual to combine empirical findings and knowledge from real processes. In general, empirical data are used to calibrate the model, and the results of the simulation process are used for planning, design and analyzing real experiments. A SPS model usually focuses on specific portions of the software development process, although it may represent the entire life cycle of the development process.

A model is at any rate an approximation and a simplification of the real system, and the model developer has to investigate in order to identify and model the aspects of the software process that are relevant to addressing the issues and questions he is studying.

III. An Extreme Programming Overview

In recent years Extreme Programming (XP) [3] is the agile methodology that has received most attention. It rejects the pomp and ceremony of traditional waterfall methodologies and the use of complex tools and solutions in favor of people doing the simplest things that could possibly work. It is defined by a set of practices that embody four fundamental values: communication, feedback, courage and simplicity. XP is based on the famous twelve practices of Planning Game, Small Releases, Metaphor, Simple Design, Testing, Refactoring, Pair Programming, Continuous Integration, Collective Code Ownership, On Site Customer, Don't Burn Out (40-Hour Work Week), and Coding Standards. These practices have been recently reviewed by the original proposer itself – Kent Beck – in the second edition of the first XP book [1].

A. More on Pair Programming

Pair Programming practice states that any production code must be created by a pair of developers working together at the same computer. Here we report some quantitative studies conducted to assess the validity and efficiency of this practice.

An experiment conducted by Nosek at Temple University studied 15 full-time, experienced programmers working for 45 minutes on a challenging problem, important to their organization, in their own environment, and with their own equipment. Five worked individually, ten worked collaboratively in five pairs, at the same conditions. The pairs completed the task 40% more quickly and effectively by producing better algorithms and code in less time [4].

Laurie Williams – University of Utah – conducted a controlled experiment on 28 students working in pair programming and 13 students in individual programming. She has found that paired programmers spent, on average, 15% more time to complete two projects than the solo programmers spent to complete one, suggesting that pair programming is 40-50% faster than solo programming. On the other hand, pair programmers produced 15% fewer defects than the individuals and implemented the same functionalities in fewer lines of code [5].

Both the above experiments and many others have found that people learn significantly more about the system and about software development, also giving better information flow and team dynamics.

IV. Related work

In spite of the great diffusion of Extreme Programming (XP) in academic and industrial field, only recently the first attempts of XP processes simulation have appeared, all using the System Dynamics approach. Here we cite some significant contribution.

In [6] Cao proposes a system dynamic simulation model to investigate the applicability and effectiveness of agile methods and to examine the impact of agile practices on project performance in terms of quality, schedule, cost, customer satisfaction.

Misic et al. [7] investigate the possibility of using system dynamics to model, and subsequently simulate and analyze, the software development process of the XP software development process. In particular, they consider the effects of four practices of this methodology: pair programming, refactoring, test-driven development, and small developmental iterations.

In [8] Kuppuswami et al. propose a system dynamics simulation model of the XP development process to show the constant nature of cost of change curve that is one of the most important claimed benefits of XP. They also describe the steps to be followed to construct a cost of change curve using the simulation model.

One of the most relevant works was perhaps made by Kuppuswami et al. [9]. They developed a system dynamics simulation model to analyze the effects of the XP practices on software development effort. The developed model was simulated for a typical XP project of the size of 50 User Stories and the effects of the individual practices were computed. The results indicated a reduction in software development cost by enhancing the usage levels of individual XP practices.

V. The Simulation Modeling Approach

We decided to implement our model following a discrete event approach. In this section we give some insights about this technique and some explanation on how our model has evolved from the initial choice.

The main reason why we chose the Discrete Event Simulation (DES) is that inside a DES model each entity is well identified and is characterized by a number of attributes whose values can be possibly changed by the execution of some specific events. So, we can inspect the status of each model entity at each time step of the simulation, giving us the possibility to better understand the evolution of the process during a simulation run.

In a DES the simulation is advanced to the time of the next significant event. The execution of each event possibly triggers the creation of other subsequent events. The approach of jumping between significant points in time is more efficient and allows models to be evaluated more quickly.

One other advantage of the DES approach is that it allows us to define a stochastic model and perform Monte Carlo simulations, taking into account of the intrinsic risk and uncertainty of real projects. For example, the estimated effort required to implement each User Story and its priority are not fixed, and should be better modelled using appropriate random distributions.

However, as we were deeply defining our model, we observed that many attributes could be better modelled with time-continuous functions. This has led us to choose a model that can be seen as an hybrid of Discrete Event and System Dynamics models.
In fact, we use integration rates (typical of System Dynamics) which updates entity attributes at time steps driven by event execution (Discrete Event). For example, the rate at which a developer implements a specific task increases with

her skill level, that will vary continuously with her cumulated experience on the project. Nevertheless, our model updates developers' experience levels at discrete steps, in correspondence of each development session closure. So, we can conclude that we are following an hybrid modeling approach (see more details in Martin and Raffo [10].)

VI. Model Description

The model is characterized by several activities (Release Planning, Development Session, etc). The inputs to these activities are entities (User Stories, Integrated code, etc) that are modified and created by other instances of activities. The class diagram in figure 1 shows the relationships among the high-level entities of the XP process model. The activities are eventually composed of sub-activities such as the *user story estimation* activity. Each activity is executed by one or more actors of the process. The identified

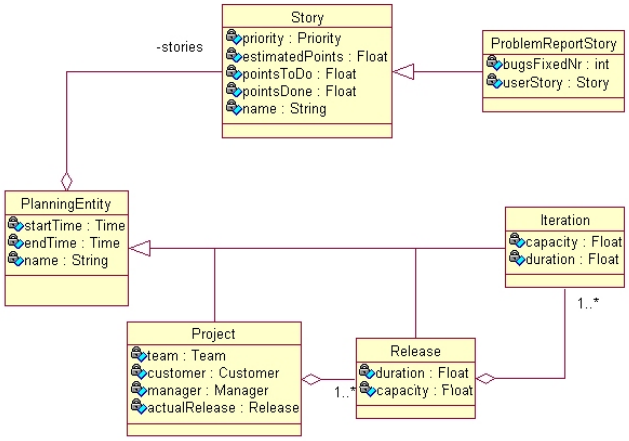


Fig. 1. *Class diagram of some of the high-level model entities.*

actors are the Team, made up of Developers, the Customer and the Manager, as shown in figure 2. Each actor has some attributes, which vary in time, and can perform a number of actions. These actions can be performed in cooperation with other actors (two developers working in pair-programming) in order to carry out a particular activity (see section VI-A.1).

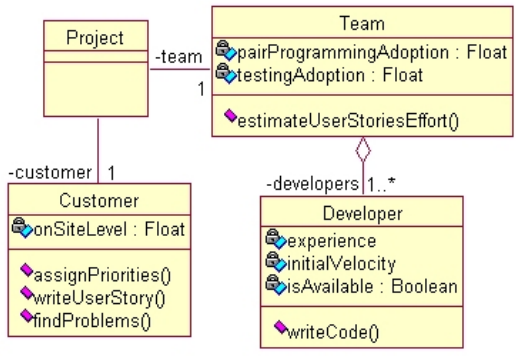


Fig. 2. *Extract from the class diagram of the identified XP actors.*

TABLE I
Input parameters and output variables of the model.

Input parameters	Output variables
Number of initial USs	Number of final USs
Number of developers	Defect density
Mean and standard deviation of initial USs estimation	Number of Classes, methods, DSIs
Initial Team velocity	
Number of Iterations per Release	⋮
Typical iteration duration	*Each modelled entity*

The time granularity of our model is that of a development session, which is typically of a couple of hours. The equations that regulate the variations on the model entities and the execution of each activity have been taken from existing models, empirical data and, where necessary, from authors assumptions. About the statistical distributions used in the model we have mostly chosen gaussian and log-normal functions.

In table I we report the input parameters needed to start a simulation and the output variables that can be obtained from the simulation. Moreover, the model is characterized by a number of inner parameters that have been mainly taken by existing models and

A. Model Dynamics

The project starts with an initial number of User Storyes (USs), which identify the main requirements of the project and represent a preliminary evaluation of the project's size. These USs are prioritized by the Customer and subsequently estimated by the Team using values taken from statistical distributions.

This estimate is affected by a stochastic error which is decreased by the overall experience of the Team on the project. When an US estimation exceeds a certain limit (a portion of the Iteration capacity) it will be splitted into two or more USs. The next phase consists of choosing the USs which will be implemented for the next Release and consequently assigned to a specific Iteration.

During an Iteration, design and development of the scheduled USs is performed. This activity produces the source code required to implement the functionality described by each US. The produced code is characterized by size (in terms of number of classes, methods and locs) and quality (number of defects).

The time actually spent to implement each US is affected by the estimation error and by the velocity of the developers who have worked on it[1]. In addition, it is influenced also by adoption levels of *Pair Programming* and *Test Driven Development (TDD)* practices.

In some cases not all the planned USs are completed within an Iteration. These USs are planned again and

[1] Developers are statistically different from each other in terms of initial skill and initial velocity. These attributes increases in accordance to the experience gained on the project.

implemented in next Iterations. Moreover, the Customer can write new USs and possibly report problems that he has found after each release of the system.

After the end of each Release the Customer can report a number of problems he/she has found on the project released up to that moment. This number is related to the defect density of the system. These reports are planned by the Team as the other USs (ProblemReportStory (PR-US)) each of which has an associated US affected by the problem founded by the Customer. The implementation of each PR-US has the effect of reducing the number of defects of the related US.

For the sake of brevity, we only report an informal description of the Development Session activity.

A.1 Development Session

During a development session, characterized by a certain duration taken from a statistical distribution, a Developer develops the code relating to a particular User Story. In an XP project the development session would be normally performed by two developers working together at a single computer (Pair Programming). However, this practice is rarely adopted completely. For this reason, our model has an input parameter called *Pair Programming Adoption* that indicates the percentage of usage of this practice. This parameter gives the probability at which a Development Session will be performed by two developers instead of one.

A Development Session performed in pair programming is more efficiently than a "solo-programming" in terms of the time needed to implement a single User Story, defects injected (due to the continuous review made by the pair developer [5]), learning efficiency (the developer skill increases faster if one works together with another developer [11], [12]).

In particular we have made the assumption that the velocity of a pair of Developers is given by the average of the velocities of the two developers involved, increased by 40%, as found in some empirical researches [5]). Moreover, in these studies it has been found that the defects injected during a pair-session is less than that of a "solo-programming". So, we model this behavior imposing that the maximum number of bugs injected during a Development Session activity is dictated by the best developer of the pair in terms of skill.

In an XP project Developers should write the code with the relating unit tests. Also in this case the model has a parameter called *Tdd Adoption* that accounts for the level of adoption of this practice. This parameter decreases the velocity of the development session and the number of the defects injected [13].

At the end of a Development Session activity, new code is produced and existing code is modified, introducing inevitably a certain number of defects. The level of these changes are affected by stochastic variables influenced by both Developers' attributes (experience and skill) and the usage levels of individual XP practices (*Testing* and *Pair Programming*).

VII. Verification and Validation of the Model

One of the major problems in process simulation is the effective calibration and validation of the developed simulator. In order to reach this goal, data sets gathered in real projects are needed. However, these data are difficult to obtain for several reasons. The greater part of real projects are developed inside privately-owned companies that, for obvious reasons, are generally reluctant to publish data regarding their inner development process.

Also, it is difficult to find companies that develop software using XP and systematically collect information regarding their development process. Moreover, in the case it happens, it is not guaranteed that the level of detail of the information collected is sufficient for a proper simulation calibration and validation. We can cite two XP projects where tracking activity has been conducted systematically and whose data are available at a sufficient level of detail: *Repo Margining System* [14] and *Market Info* [15].

In order to calibrate the parameters of the simulation model, we have used some input variables (see table I) coming from the *Repo Margining System* project [14], such as the number of developers, the release duration and so on. Also, we have used the project and process data gathered during the first iteration. We then simulated the evolution of the project starting from the second iteration.

With these input parameters a number of simulation runs have been performed. Then, we have iteratively calibrated the model parameters in order to better fit the final results of the real project. In table II the simulation outputs are compared with the ones taken from the *Repo Margining System* case study.

TABLE II

Comparison between simulation results averaged on 200 runs and the Repo Margining System case study. Standard deviations are reported in parenthesiss. A story point corresponds to 30 minutes of work.

Output variable	Simulation	Real Project
Total days of Development	60,3 (22,8)	60
Number of User Stories	28,9 (7,45)	29
Estimated Effort [Story points]	478,2 (183,3)	474
Actual Effort [Story points]	811,3 (296,4)	793
Number of Releases	2,4 (0,8)	2
Iterations per Release	2,7 (0,3)	3
Developed Classes	245 (108,2)	251
Developed Methods	1073 (475)	1056
DSI	15646 (6922)	15543

A conceptual model validation has been done interviewing some individuals familiar with the XP process itself. The proposed approach was presented, and its various concepts – roles, activities and artifacts – were explained in detail. The collected feedback on our approach was positive.

Also, we performed an event validation process comparing the sequence of the events produced by the simulation

with those of a real XP process.

As regards the verification of the correctness of the simulator, we implemented the system using pair-programming. Following this practice, a continuous review was made by the pair-developer diminishing, in this way, the probability of introducing errors during the implementation of the simulator. In addition, we covered all the functionalities implemented with unit and acceptance tests, enabling an automatic and continuous verification of the correctness of the system.

VIII. Results

We are developing the simulator following the XP methodology and using XPSwiki [16], that is an XP project management tool developed by our research group, in order to plan and monitor the process development.

In the first release of our project, the simulator allowed to simulate the Release and Iteration Planning activities and the code production phase where an user story is converted into production code by the Team. In the second development release we implemented the possibility to vary the level of usage of Pair Programming from maximum (100% = *Fully Adopted*) to minimum level (0% = *Not adopted*), while in the third release we introduced the influence of Test Driven Development. In next releases we will model in more detail the already implemented system and the influence of other XP practices.

The present model has been calibrated using data from a real project. The case study we have chosen is Repo Margining System (see section VII), which has been performed following the XP process. In this research we investigate the following question:

> *How would have been the simulated project if the team had not followed the Pair Programming (PP) practice at all?*

According to what we have found in literature (see section III-A) we have formulated our hypotheses:

Hypothesis A: The residual defect density of the project using the PP practice is different from that obtained without PP ($H_0 : \overline{d}_{noPP} = \overline{d}_{PP}$, $H_1 : \overline{d}_{noPP} \neq \overline{d}_{PP}$).

Hypothesis B: The number of working days needed to complete the same number of functionalities using PP is different from that without PP ($H_0 : \overline{t}_{noPP} = \overline{t}_{PP}$, $H_1 : \overline{t}_{noPP} \neq \overline{t}_{PP}$).

Hypothesis C: The number of lines of code needed to implement the same number of functionalities using PP is different from that without PP ($H_0 : \overline{DSI}_{noPP} = \overline{DSI}_{PP}$, $H_1 : \overline{DSI}_{noPP} \neq \overline{DSI}_{PP}$).

We have inspected the two project conditions (PP= 0% and PP= 100%) in terms of total days needed to complete the same number of functionalities, residual defect density and final Delivered Source Instructions (DSI). For each of the conditions we have performed 200 simulation runs. Results are reported in table III.

The results with PP = 100% (2nd column) are the same as those reported in table II, which have been obtained simulating the Repo Margining System project.

TABLE III

Comparison between simulation results averaged on 200 runs obtained varying the usage level of PP (TDD level = 100%). Standard deviations are reported in parenthesis.

Output variable	PP level = 0% (TDD = 100%)	PP level = 100% (TDD = 100%)
Working days	51,1 (23,6)	60,3 (22,8)
Released USs	28,7 (7,6)	28,9 (7,5)
Defects/KDSI	23,0 (5,3)	19,7 (4,5)
KDSI	21,5 (10,2)	15,6 (6,9)

Looking at the 1st column (table III), it can be seen how some outputs of the same simulated project have varied only excluding the use of Pair Programming. We have found that not using Pair Programming at all, the duration of the project (in terms of working days) decreases by 15%. The number of User Stories remains quite the same in both cases. Instead, the defect density increases by 17% when we exclude the use of PP. In addition, the use of PP decreases the number of DSI by 27%.

Consequently, we can say that the use of the Pair Programming practice increases the total cost of development (working days), but it is repaid by an increase of the quality of the project (in terms of defect density) and by a better design, in terms of fewer lines of code per User story (0,75 KDSI/US against 0,54 KDSI/US).

These results are quite in agreement with those found in [5] and [4], as previously described in section III-A.

In addition we have performed a two-sided t-test ($\alpha = 0.05$). The test results (table IV) have confirmed our hypotheses (A, B and C) with a statistical significance of 95%.

TABLE IV

Results of the two-sided t-test ($\alpha = 0.05$) of the two samples obtained with PP=0% and PP=100% (TDD=100%).

	t_{score}	t_{crit}	H_0	$P-value$
Hyp_A	6,87	1.97	rejected	0.00
Hyp_B	3,98	1.97	rejected	0.00
Hyp_C	6,75	1.97	rejected	0.00

We have also performed the same experiment varying the adoption of Pair Programming when TDD is not used.

We have found (table V) that not using Pair Programming at all, the duration of the project decreases by 12%. The number of User Stories remains quite the same in both cases. Instead, the defect density increases by 16% when we exclude the use of PP. In addition, the use of PP decreases the number of DSI by 28%.

Analyzing the first column of the tables III and V we can see how keeping the usage level of PP constant and varying from 0% to 100% the usage level of TDD, the working days increases by 14%, the defects density decreases by 18%,

TABLE V

COMPARISON BETWEEN SIMULATION RESULTS AVERAGED ON 200 RUNS OBTAINED VARYING THE USAGE LEVEL OF PP (TDD LEVEL = 0%). STANDARD DEVIATIONS ARE REPORTED IN PARENTHESIS.

Output variable	PP level = 0% (TDD = 0%)	PP level = 100% (TDD = 0%)
Working days	45,0 (23,2)	51,1 (19,1)
Released USs	28,8 (7,9)	28,8 (7,6)
Defects/KDSI	28,0 (5,3)	24.1 (6,0)
KDSI	18,0 (8,2)	13,0 (6,1)

while the number of User Stories remains quite the same.

Other interesting results can be observed looking at the 2nd column of both tables. Starting from the Repo Margining results (PP=100% and TDD =100%), we have excluded the use of TDD. It can be noticed that the use of TDD increases the duration of the project by 18%. In other words, the Repo Margining project would have been concluded 15% of the time before the actual end time if TDD was not used. The extra time taken by TDD could be attributed to the time needed to develop test cases. In addition, test cases lead to an increment of the source instructions of the project, as confirmed by the simulation results (+ 20%). On the other hand, the simulation produced a decrease in defect density by 18% when TDD is used. These figures are quite in agreement to those found empirically in [17].

The extreme case is when we simultaneously vary the usage of both the practices. From TDD=PP=0% to TDD=PP=100% we have obtained an increase in time by 34%, a decrease by 30% of the defect density and a decrease in DSIs by 13%. However, these results are not confirmed by any empirical findings because we haven't found related studies in literature, so we cannot validate this last experiment.

All the results illustrated here have shown a statistical significance ($\alpha = 0,05$) after having performed a two-sided t-test.

IX. Conclusions and Future Work

In this paper we have presented a simulation model of XP process that we have developed in order to evaluate the effectiveness of this methodology.

We have calibrated our model using some results obtained from a real project. Then we have simulated the variation of the usage level of two keys XP practices: Pair Programming and Test Driven Development.

We have found that increasing the usage of such practices the defect density of the project significantly decreases. On the other hand, the results have shown an increase on the number of days needed to implement the same functionalities.

Let us note that our model is not a complete representation of the intrinsic complexity of these practices and of the development process itself. We have based our model on what has been empirically found up to now, but many other issues have to be better investigated.

We are planning to improve the current simulator modeling other practices and activities of the software development process, with emphasis on XP. Another important stage of our research will be the validation of the model using other real projects and experiments.

X. Acknowledgements

This work was supported by MAPS (Agile Methodologies for Software Production) research project, contract/grant sponsor: FIRB research fund of MIUR, contract/grant number: RBNE01JRK8.

References

[1] Kent Beck and Cynthia Andres, *Extreme Programming Explained: Embrace Change- Second Edition*, Addison-Wesley, 2004.

[2] Marc I. Kellner, Raymond J. Madachy, and David M. Raffo, "Software process simulation modeling: Why? What? How?," *The Journal of Systems and Software*, vol. 46, no. 2–3, pp. 91–105, Apr. 1999.

[3] Kent Beck, *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 1999.

[4] John T. Nosek, "The case for collaborative programming," *Commun. ACM*, vol. 41, no. 3, pp. 105–108, 1998.

[5] Alistair Cockburn and Laurie Williams, "The costs and benefits of pair programming," in *Proceedings of the First International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP2000)*, Cagliari, Sardinia, Italy, June 2000.

[6] Lan Cao, "A modeling dynamics of agile software development," in *Companion of 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*. 2004, pp. 46–47, ACM Press.

[7] Vojislav B. Misic, Hudson Gevaert, and Michael Rennie, "Extreme dynamics: Modeling the extreme programming software development process," in *Proceedings of ProSim04 workshop on Software Process Simulation and Modeling*, 2004, pp. 237–242.

[8] S. Kuppuswami, K. Vivekanandan, and Paul Rodrigues, "A system dynamics simulation model to find the effects of xp on cost of change curve.," in *XP2003 Conference Proceedings*, 2003, pp. 54–62.

[9] S. Kuppuswami, K. Vivekanandan, Prakash Ramaswamy, and Paul Rodrigues, "The effects of individual xp practices on software development effort," *SIGSOFT Softw. Eng. Notes*, vol. 28, no. 6, pp. 6–6, 2003.

[10] Robert H. Martin and David Raffo, "A model of the software development process using both continuous and discrete models.," *Software Process: Improvement and Practice*, vol. 5, no. 2-3, pp. 147–157, 2000.

[11] K. Vivekanandan, *The Effects of Extreme Programming on Productivity, Cost of Change and Learning Efficiency*, Ph.D. thesis, Doctor of Philosophy in Computer Science and Engineering, 2004.

[12] D. Sanders, "Student perceptions of the suitability of extreme and pair programming," in *Proceedings of XP Universe Conference*, Raleigh, NC, 2001.

[13] Ivana Turnu, Marco Melis, Alessandra Cau, Michele Marchesi, and Alessio Setzu, "Introducing TDD on a free-libre open source software project: a simulation experiment," in *Proceedings of Qute Swap workshop on QUantitative TEchniques for SoftWare Agile Processes*, 2004.

[14] Klondike Team, "Tracking-aworkingexperience," Published on: http://www.communications.xplabs.com/paper2001-2.html, 1900.

[15] PierGiuliano Bossi, "Extreme programming applied: a case in the private banking domain," in *Proceedings of OOP2003*, 2003.

[16] Sandro Pinna, Simone Mauri, Paolo Lorrai, Michele Marchesi, and Nicola Serra, "XPSwiki: an Agile Tool Supporting the Planning Game," in *XP2003 Conference Proceedings*, 2003, pp. 104–113.

[17] Boby George and Laurie Williams, "An initial investigation of test driven development in industry," in *Proceedings of the 2003 ACM symposium on Applied computing*. 2003, pp. 1135–1139, ACM Press.

Section III

Process Simulation Modeling – Focus on Methodology

DEVS-based Software Process Simulation Modeling: Formally Specified, Modularized, and Extensible SPSM *

KeungSik Choi, Doo-Hwan Bae, TagGon Kim
Department of EECS,
Korea Advanced Institute of Science and Technology, Daejon 305-701, Korea
{*kschoi, bae*}*@se.kaist.ac.kr, tkim@ee.kaist.ac.kr*

Abstract

This paper proposes DEVS (Discrete Event System Specification)-based software process simulation modeling method which is a formally specified, modularized, and extensible simulation modeling approach. The proposed approach adopts DEVS formalism, a general purpose discrete event modeling and simulation framework, to the software process simulation modeling domain. This approach enables us to clearly understand the software process simulation model by formal specification, and provides explicit extension point to extend the simulation model for a specific purpose.

This approach also provides naturally hybrid software process simulation modeling environment, which embeds DESS (Differential Equation System Specification) and DTSS (Discrete Time System Specification) into DEVS formalism. This hybrid approach overcomes some limitations of system dynamics simulation models, such as difficulties of controlling process execution, representing explicit process activity, and modeling inherent uncertainty.

1. Introduction

Many researchers have investigated why Software Process Simulation Models (SPSMs) are not widely used in industry and proposed several approaches. Raffo [1] argued that industry is not using SPSMs to their full advantages, because process models are difficult to build and maintain. As a solution, he proposed designing a Generalized Software Process Simulation Model (GPSM) that can be tailored quickly to a particular project scenario and deployed rapidly. He also reviewed three possible research areas applicable to GPSM: Modularization, Software Product-Line, and Cognitive Pattern. Angkasaputra and Pfahl [2] proposed to make SPSMs more agile by taking benefits from agile methods and design patterns to reduce product delivery time and budget. Ruiz [3] developed a Reduced Dynamic Model (RDM) which simplified Abdel-Hamid and Madnick's model [4] to easily learn and understand the model and to use the model at the initial phases of a project where the available or known information about the project is little.

Although these approaches alleviate some of the difficulties in using SPSMs in industry, there are other obstacles to apply SPSMs, especially system dynamics models, in software development projects. Traditional SPSMs are difficult to understand and extend, because most of researches concentrate on developing simulation models and displaying simulation results. For example, the stock flow diagrams of system dynamics models have complicated arrows, circles, rectangles, etc., which make it difficult to understand the process activity controls and variable interactions. Furthermore, those models don't have clear specifications and are rarely verified. Because simulation models are also software systems, we need a clear specification of the simulation model. Liu [5] claimed that to improve the maintainability we have to enhance the structural and behavioral specifications.

In an attempt to answer the aforementioned arguments we propose a DEVS (Discrete Event System Specification) [6]-based software process simulation modeling technique which is a formally specified, modularized, and extensible simulation modeling approach. With this approach, we can clearly specify and verify the simulation model and extend the model using Object-Oriented framework provided by the DEVS simulation engine [7]. Another benefit of this approach is this technique can solve the disadvantages of system dynamics models such as difficulties in controlling activity sequencing, describing discrete process steps, representing attributes of individual entities, and modeling un-

* This work was supported by the Ministry of Information & Communication, Korea, under the Information Technology Research Center (ITRC) Support Program.

certainties [11].

The structure of this paper is as follows. In Section 2, we briefly introduce the DEVS formalism and the simulation environment. Section 3 describes the proposed DEVS-based software process simulation modeling method and demonstrates how to extend and tailor the simulation model. Section 4 compares existing hybrid simulation approaches and proposed DEVS-based SPSM in various aspects. Section 5 summarizes the main results of this paper and gives a plan for future work.

2. Background

2.1. DEVS formalism

DEVS is a general formalism for discrete event system modeling based on set theory [6]. It allows representing any system by three sets and four functions: Input Set, Output Set, State Set, External Transition Function, Internal Transition Function, Output Function, and Time Advanced Function. DEVS formalism provides the framework for information modeling which gives several advantages to analyze and design complex systems: Completeness, Verifiability, Extensibility, and Maintainability [7]. DEVS can also approximate continuous systems using numerical integration methods. Thus, simulation tools based on DEVS are potentially more general than other tools including continuous simulation tools [8]. With those properties, we applied DEVS formalism to software process simulation modeling.

DEVS has two kinds of models to represent systems. One is an atomic model and the other is a coupled model which can specify complex systems in a hierarchical way [6]. A DEVS model processes an input event based on its state and condition, and it generates an output event and changes its state. Finally, it sets the time during which the model can stay in that state. An atomic DEVS model is defined by the following structure [6]:

$$M = \langle X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta \rangle$$

where:

- X is the set of input values,
- Y is the set of output values,
- S is the set of states,
- $\delta_{ext} : Q \times X \rightarrow S$ is the external transition function, where $Q = \{(s, e) | s \in S, 0 \leq e \leq ta(s)\}$ is the total state set, e is the time elapsed since last transition
- $\delta_{int} : S \rightarrow S$ is the internal transition function,
- $\lambda : S \rightarrow Y$ is the output function,
- $ta : S \rightarrow R^{+}_{0,\infty}$ is the set positive reals bet. 0 and ∞

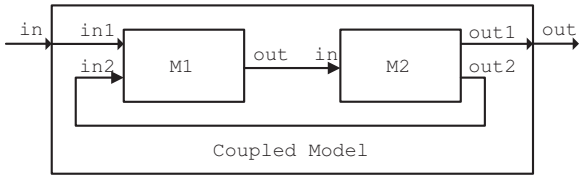


Figure 1. Coupled DEVS model

The behaviors represented by four functions of the atomic model are as follows:

- An atomic model can stay only in one state at any time
- The maximum time to stay in one state without external event is determined by $ta(s)$ function
- When an atomic model is in a state $(0 \leq e \leq ta(s))$, it changes its state by δ_{ext} function if it gets an external event
- If possible remaining time in one state is passed $(e = ta(s))$, it generates output by λ function and changes the state by δ_{int} function

DEVS coupled model is constructed by coupling DEVS models. Through the coupling, the output events of one model are converted into input events of other models. In DEVS theory, the coupling of DEVS models defines new DEVS models (i.e., DEVS is closed under coupling) and then complex systems can be represented by DEVS in a hierarchical way [6]. Figure 1 shows a coupled DEVS model. M1 and M2 are DEVS models. The M1 model has two input ports (”in1” and ”in2”) and one output port (”out”). The M2 model has one input port (”in1”) and two output ports (”out1” and ”out2”). They are connected by input and output ports (e.g., ”out” port of M1 is connected to ”in” port of M2) internally, which is called Internal Coupling (IC). The M1 model is connected by external input, ”in” port of Coupled Model, to ”in1” port, which is called External Input Coupling (EIC). The M2 model is connected to output port ”out” of Coupled Model (e.g., ”out1” port of M2 is connected to ”out” port of Coupled Model), which is called External Output Coupling (EOC). According to the closure property, the coupled model in Figure 1 can be used as a DEVS model and it can be coupled with other DEVS models.

2.2. Simulation environment

The DEVSimHLA [7] is a C++ based DEVS simulation environment which is integrated with Microsoft Visual Studio .NET. It, therefore, provides the advantages of Object-oriented framework such as encapsulation, inheritance, etc. The DEVSimHLA coordinates the event sched-

ules of atomic models in a system and provides classes and APIs for simulation.

With DEVS formalism and DEVSimHLA, we can get several advantages. First, we can specify the systems mathematically which gives straight forward verification method. Second, we can model hierarchical and modularized systems which enhance understandability and extensibility. Third, we can reuse simulation models by inheritance.

3. DEVS-based software process simulation modeling approach

This section describes how to model software development process using DEVS formalism. We develop a generic and simplified software process model to estimate the cost and duration of a project based on the initial information on the project such as project size and project duration. We will show the overall architecture of the proposed simulation model, and how to formally specify the software process simulation model using DEVS formalism. We also demonstrate how to extend and tailor the DEVS-based software process simulation model for a specific purpose. Finally, we will emphasize how our approach provides naturally hybrid simulation environment.

3.1. Overview

The purpose of this simulation modeling approach is to develop a formally specified, modularized, and extensible simulation model to make full advantages of SPSMs. We analyze published system dynamics models through literatures and books, and identify several limitations as follows [11]:

- Understanding and maintaining simulation models is difficult because of no clear specification
- Extending simulation models is difficult because of no explicit reuse mechanism and extending points provided
- Describing process steps is difficult because of no explicit mechanism to control the activity sequences
- Representing error prone modules or variable productivity of developers is difficult because of no individual entities and entity attributes
- Modeling uncertainties inherent in estimates of model parameters is difficult

We referenced system dynamics models provided by Vensim simulation tool [9] and Abdel-Hamid and Madnick [4], and simplified it for our purpose. We assume that we already analyzed all the dynamic interactions of software process variables using such as Causal-Loop Diagram (CLD).

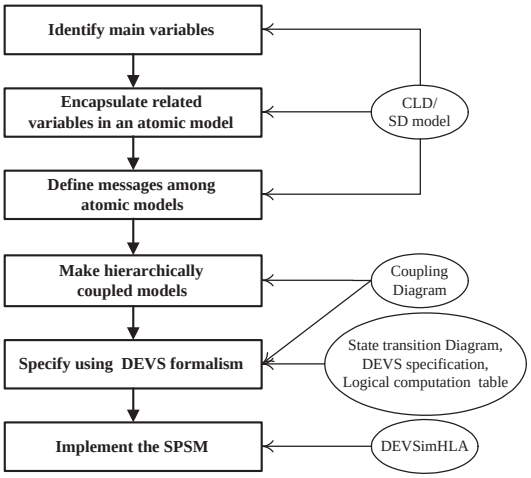


Figure 2. Overall approach of DEVS-based software process simulation modeling

Figure 2 shows our approach of proposed DEVS-based software process simulation modeling. First, we identify main variables which drive the simulation result variables such as an effort and a duration in software process simulation models. For example, we identified that the most important variable is a *workflow* rate which is determined by many factors such as productivity, total workforce, synergy, and communication overhead. We then construct an atomic model by encapsulating closely related variables centered on the main variable identified in the previous step, and identify interaction points (variables) among atomic models. There are two kinds of interaction points. One is a variable which is calculated by other atomic models and updated by input messages. The other is a variable which is computed in this model and becoming an output message. After constructing atomic models and interaction points, we define messages which transmit simulation information among atomic models. We then couple the models hierarchically. Hierarchical modeling enables us to modularize the simulation model and makes it simple and understandable.

When atomic and coupled models are defined, and their interactions are identified, we specify the simulation models using four methods which include a coupling diagram, a state transition diagram, DEVS specification, and a logical computation table. These methods we propose provide a clear, verifiable, understandable, and extensible specification for the software process simulation model. Finally, we implement the specification using the DEVSimHLA environment [7].

3.2. Overall architecture of the DEVS-based software process simulation model

Figure 3 illustrates the overall architecture of the simulation model, which is composed of a DevelopmentPhase and an ExperimentalFrame model. It shows the most basic structure of the software development project. We design this to make the model simple and easy to extend. The DevelopmentPhase model can represent the whole software development life-cycle and also can represent each phase of the software development life-cycle. The ExperimentalFrame model can be reused and extended for a specific simulation purpose.

The DevelopmentPhase model represents any phase of the software development life-cycle. It can be extended to any of the life-cycle such as requirements, design, or implementation. It also can be a Waterfall or Incremental life-cycle by coupling the DevelopmentPhase model each other. The WorkToBeDone, WorkDone, and Rework models are like a level variable in stock flow diagram. The WorkToBeDone model stores the amount of work to be done and sends it to the Work model. The WorkDone model integrates the workflow rate to compute the work done, and the Rework model computes the amount of rework to do again and sends the rework rate to the WorkToBeDone model.

The ExperimentalFrame model plays a role of a measurement system or observer like an oscilloscope in electronics. It generates inputs to the observed system, and accepts and analyzes the experiment data. It is a system that interacts with the system of interest to obtain the data of interest under specified conditions. In this simulation model, it generates WorkToDo message, which is an initial work to do. The WorkToDo message will be processed by the DevelopmentPhase model and sends the simulation data, as a WorkMonitoring message, to the ExperimentalFrame model.

The TimeIntervalGenerator model in the ExperimentalFrame is an executive which drive the simulation execution. It generates a Time event in a small-enough constant-time interval to make the WorkToBeDone model change its state and generate the WorkToDo message. The TimeIntervalGenerator model enables the project variables to dynamically interact each other, which allows this model to become a naturally hybrid simulation model. The naturally hybrid simulation approach will be discussed in Section 4

The WorkMonitoring message contains simulation variables which are stored and analyzed by the SimAnalysis model in the ExperimentalFrame model. This message includes level variables, rate variables, and auxiliary variables in system dynamics representation. These variables are dynamically updated through the feedback loop, which shows the effects of complex dynamic software development process.

The Done message makes this simulation model stop. The WorkToBeDone model generates the Done message when the work is done. The interaction point of the DevelopmentPhase model (e.g., Time, Done, and WorkMonitoring) depicted in a small rectangle in Figure 3 is called an input or an output port. This is an explicit extension point in DEVS-based software process simulation model. For example, we can instantiate the DevelopmentPhase model to a Requirements model and a Design model, and then connect the Done port of the Requirements model to the WorkIn port of the Design model. The model extension issues will be further discussed in Section 3.4.

3.3. Formal specification using DEVS formalism

As we mentioned before, we use four methods (coupling diagram, state transition diagram, DEVS specification, and logical computation table) to specify software process simulation model. Using these methods we can develop a clear, verifiable, understandable, and extensible specification for a software process simulation model.

The coupling diagram shows the internal structure of the coupled model and the flow of simulation data. The state transition diagram specifies how the state changes as the system responds to its different inputs. This is the explicit mechanism to control the activity sequences. Based on the input and the state of the model we can change the behavior of the model. The DEVS specification defines the behavior of the model with 3 state sets and 4 functions. This provides the sound framework to verify the behavior of the simulation model, because the formal specification gives clear interpretation of that. The logical computation table describes how to calculate the project variables when the model changes its state by external input or internal time expiration.

Figure 4 shows the coupling diagram of the Work_Coupled model which shows the internal structure of the Work model in Figure 3. The Work_Coupled model comprises ScheduleMng, WorkforceMng, and WorkflowRateMng model. The ScheduleMng model calculates the remained schedule time to complete the job and based on this it calculates the required workflow. The required workflow can be KLOC/day or FP (Function Point)/day, etc. The required workflow is sent to the WorkforceMng model which returns the indicated workforce. The indicated workforce means required workers to complete the job based on the schedule. Based on this indicated workforce, the ScheduleMng model determines the amount of overwork. The WorkforceMng sends the total workforce and the WorkflowRateMng model calculates the workflow rate and work quality. The workflow rate means work done per day (e.g., KLOC/day or FP/day).

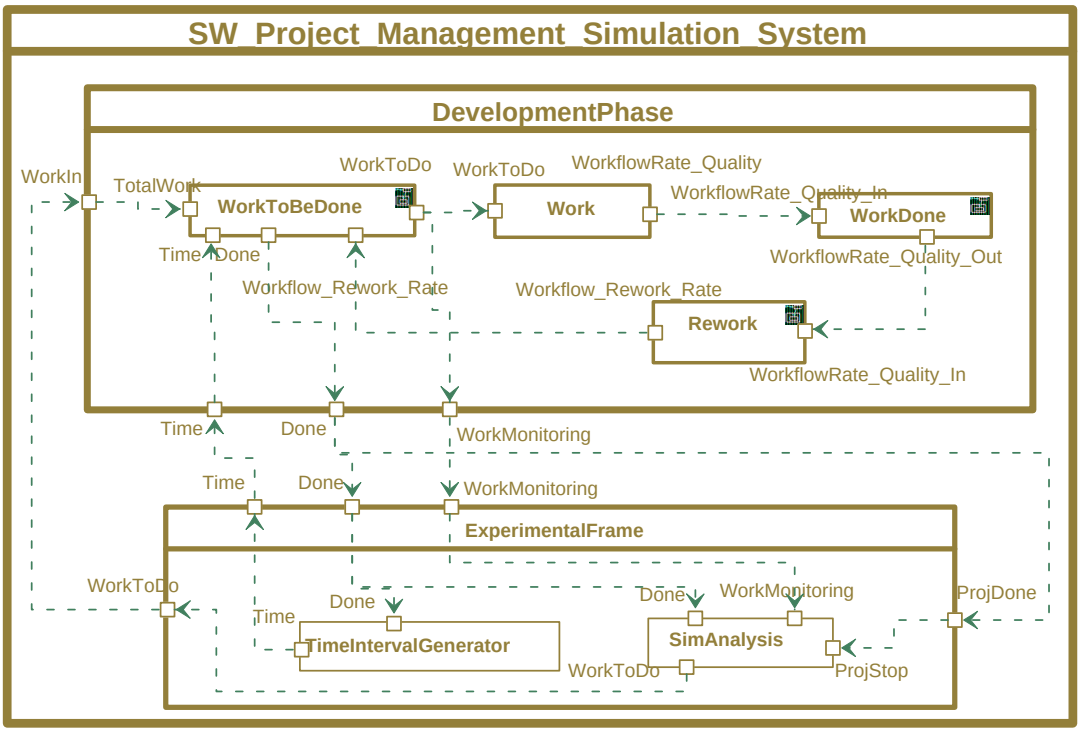


Figure 3. Overall architecture of DEVS-based SPSM

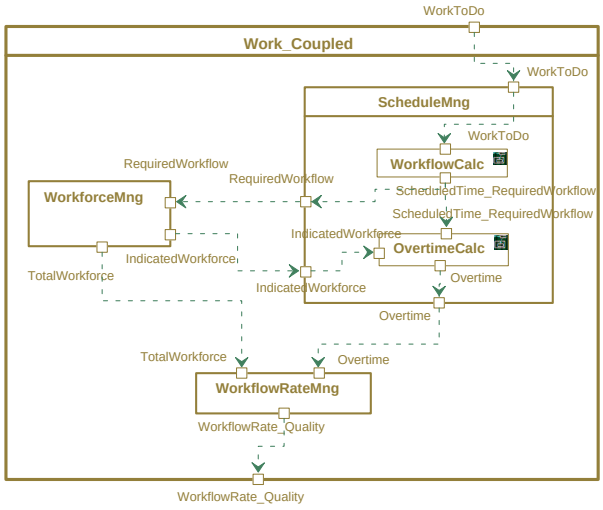


Figure 4. Coupling diagram of Work coupled model

Figure 5 depicts the state transition diagram of the OvertimeCalc atomic model. This diagram shows that the initial state is a Wait state and it stays infinitely until it receives external input message. The model changes the state to the GetIndicatedWorkforce state when it receives external input message, ScheduleTime_RequiredWorkflow, and then the model changes the state to the OvertimeCalc state in response to the IndicatedWorkforce input. In the OvertimeCalc state, the model generates the Overtime output message and changes its state to the Wait state with 0 second delay.

Table 1 shows the DEVS specification of the OvertimeCalc model. It represents the same information with the state transition diagram but defines it more declaratively.

Table 2 shows the logical computation table of the OvertimeCalc model. This table specifies the type of the variables and the equations. When this model receives a ScheduledTime_RequiredWorkflow message in the Wait state, it saves the input variables, and when it receives an IndicatedWorkforce message in the GetIndicatedWorkforce state, it calculates the NormalWorkflow, SchedulePressure, and Overtime variables. When this model receives the external inputs again by the feedback structure after a constant-time interval (e.g., one hour or one day), the variables are dynamically recalculated. Through this mechanism, we imple-

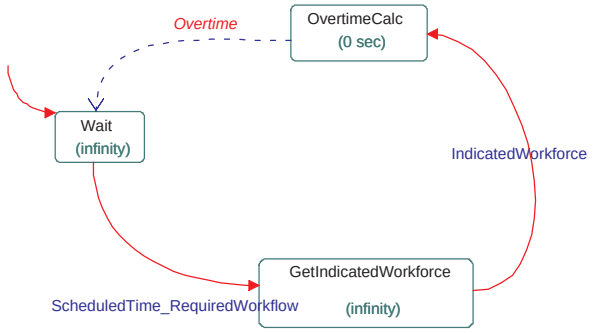


Figure 5. State transition diagram of OvertimeCalc model

ment feedback loop of the system dynamics. We don't specify the detailed variable types and equations in Table 2 because they are not our focus in this paper.

3.4. Extending and tailoring the simulation model

The model in Figure 3 is a generic and simplified one. It can estimate the effort and duration for intermediate size (21.3 Man-Month, 8 Months) [10] project. Based on this simulation model, we can extend this model rapidly and with low cost, because the base model has a clearly verified specification and explicit interfaces (ports or extension points) for extension.

For example, Figure 6 extends the base model to Waterfall life-cycle model by coupling the DevelopmentPhase model to each other. The Waterfall model starts when it receives WorkIn input which is the initial project estimation size and it ends when it receives the ProjDone message. The Requirements phase model does the job and the model outputs the Done message when it completes the job, and this message is becoming an input message of the Design phase model. Of course, we have to modify the variables and the dynamic equations of each model to apply the characteristics of each phase. The ExperimentalFrame model is also reused and extended to store and analyze the project information of each phase.

Another example is shown in Figure 7. In this example, we add a ResourcePool model which manages shared human resources. If one organization has limited resources and performs two projects simultaneously, we suffer resource conflicts. Proj_A and Proj_B are modified version of the DevelopmentPhase model in Figure 3. The WorkforceMng model in the Proj_A is modified to request new workers to the organization's shared resource pool and it receives allocated workers. We reuse other models as it is except the WorkforceMng model. The ExperimentalFrame model is extended to store and analyze the project information of two projects.

State transition function	Logical description
δ_{ext}(Wait, Scheduled-Time_RequiredWorkflow)	Save ScheduledTime and RequiredWorkflow
δ_{ext}(GetIndicatedWorkforce, IndicatedWorkforce)	Calculate NormalWorkflow, SchedulePressure, Overtime

Table 2. Logical computation table of OvertimeCalc model

OvertimeCalc = $\langle X, Y, S, \delta_{ext}, \delta_{int}, \lambda, ta \rangle$

X={ScheduledTime_RequiredWorkflow, IndicatedWorkforce}
Y={Overtime}
S={Wait, GetIndicatedWorkforce, Overtime_Calc}
δ_{ext}(Wait, ScheduledTime_RequiredWorkflow) = GetIndicatedWorkforce
δ_{ext}(GetIndicatedWorkforce, IndicatedWorkforce)= Overtime_Calc
δ_{int}(Overtime_Calc) = Wait
λ(Overtime_Calc) = Overtime
ta(Wait) = ta(GetIndicatedWorkforce) = Infinity
ta(Overtime_Calc) = 0

Table 1. DEVS specification of OvertimeCalc model

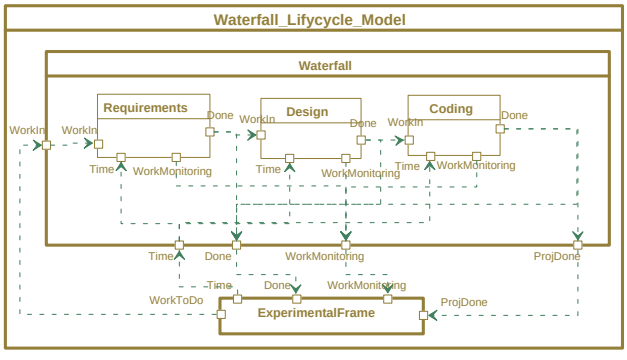


Figure 6. Extended model: Waterfall model

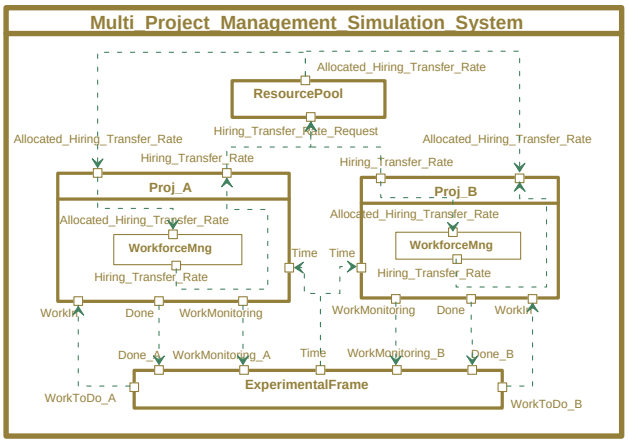


Figure 7. Extended model: Multi-project model

4. Naturally hybrid simulation modeling approach

We described DEVS-based software process simulation modeling approach, which overcomes some limitations of published system dynamics models: difficulties in understanding and extending simulation models, difficulties in describing discrete process steps, difficulties in representing individual entities and entity attributes of a variable, and difficulties in modeling uncertainties. The four methods for specifying software process simulation model provide a clear and understandable specification, and we can extend the simulation model through explicit extension points (ports). We also showed how to model discrete process steps in Section 3.4. Moreover, we can easily represent individual entities and entity attributes, and uncertainties because the proposed simulation model is basically a discrete event simulation model.

Because our approach incorporates feedback loop mechanism of system dynamics and represents discrete event, the DEVS-based software process simulation modeling technique is a hybrid simulation modeling approach. We named this approach to ”Naturally hybrid simulation modeling approach” because DEVS formalism can naturally embed Discrete Time System Specification (DTSS) and Differential Equation System Specification (DESS), which will be discussed further in this section.

4.1. Characteristics of DEVS-based naturally hybrid simulation modeling approach

In this section, we provide theoretical backgrounds on how to model continuous systems in DEVS representation,

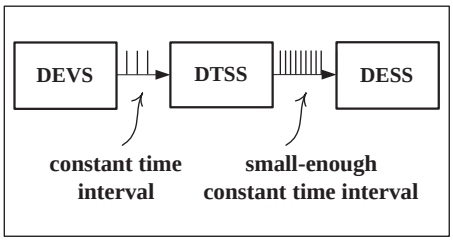


Figure 8. Formalisms embedding

and describes the characteristics of DEVS-based naturally hybrid simulation modeling approach.

Traditionally, differential equations have been solved with numerical integration in discrete time. In formal terms, Differential Equation System Specification has been embedded into Discrete Time System Specification [6]. The word ”embedded” in this context means that any system in DESS can be simulated by DTSS. Of course, errors may be introduced in the DTSS approximation of the DESS model, but it is tolerable if the discrete time is small enough. Moreover, any DTSS can be simulated by the DEVS by constraining the time advance to be constant. Therefore, if we constrain the time advance of the DEVS to be small-enough constant-time, we can model DESS with DEVS formalism. This formalism embedding illustrated in Figure 8 makes DEVS-based software process simulation modeling technique to be a naturally hybrid simulation modeling approach. We coined ”naturally hybrid” because this simulation modeling environment naturally includes both discrete and continuous simulation modeling techniques.

We also can easily model uncertainties in model parameters. For example, if we want to make a stochastic human resource transfer model, we can extend the DEVS formalism to a stochastic DEVS. The output function of DEVS, λ, can be extended like this:

$$\tilde{\lambda} : S \times [0,1] \rightarrow Y$$

Therefore, the output is characterized by a possibility distribution of $[0,1]$. We can apply this to the external transition function and internal transition function as well.

4.2. Comparisons of published hybrid simulation approaches and DEVS-based SPSM

System dynamics models describe the interaction between project factors, but do not easily represent discrete process steps. On the other hand, discrete event models may not have enough events to represent feedback loops accurately [11]. Many researches, therefore, have tried to combine discrete event simulation and continuous simulation to

Author	Martin [11]	Lakey [12]	DEVS-based SPSM
Application	Evaluate potential process changes	Project estimation & Project management	Project estimation & project management
Implementation tool	Extend	Extend	DEVSimHLA
Basic approach	Discrete event models are combined in system dynamics framework	Feedback mechanisms are approximated in discrete process	System dynamics models are implemented with DEVS which embeds DESS
	Process activities: DES	Process activities: DES	Process activities: DES
	Project environment: SD	Divide a discrete activity into multiple sub-activities and iterate multiple times	Implement feedback mechanism by constraining time advance into small-enough constant-time interval
Discrete event model	Calculates duration, total effort, errors which are passed out to system dynamics model	Product size and quality are passed on to next activity	Information on the work, process control events, and simulation results data are passed on to next activity
Continuous model	System dynamics model (human resource, manpower allocation, productivity) passes out the data to discrete event model	Each sub-activity (feedback loops of development, review, rework) iterates multiple times to incorporate dynamic feedback loops while calculating a number of equations for product, process, project factors	Workflow, human resource, manpower allocation, productivity, work quality, etc. are dynamically calculated
Limitations	Duration time of each activity can not be dynamically calculated	Coarse approximation of system dynamics model	No limitation
Timing issues	Each activity computes the duration time but advances the clock only by the specified delta time	Time advance doesn't mean anything, schedule model in each activity calculates calendar weeks	Time advance is constrained to small-enough constant-time interval, duration time of each activity is dynamically calculated

Table 3. Comparison of hybrid simulation modeling approaches

model software process more realistically. Table 3 compares existing hybrid simulation approaches with our approach.

Martin et al. [11] develop a combined model that represents the software development process as a series of discrete process steps executed in a continuously varying project environment to evaluate the potential process changes. The process activities are represented in discrete event model and the project environments, such as human resource, manpower allocation, and productivity, are modeled in system dynamics. In their approach, the discrete event models are combined in the system dynamics framework as shown in Figure 9. To support hybrid model, they advance the clock by the specified delta time while preserving the discrete scheduling time computed by discrete event model. At each delta time increment, they integrate all necessary equations until the next scheduled event time is reached. The limitation of this approach is that the duration time of each activity is not calculated dynamically. In most system dynamics approach, the activity time is dynamically influenced by many factors such as pressure, fatigue, and workforce, but this approach calculates the duration time of each activity once before executing the activity.

Lakey [12] proposes a hybrid simulation model for project estimation and management. The feedback loops are incorporated in the discrete event process by dividing a single discrete event process block into multiple iterations so that certain factors are allowed to change several times within the discrete activity. The development process has discrete four major activities as shown in Figure 9, and each of the activity incorporates four discrete sub-blocks which represent feedback loops of system dy-

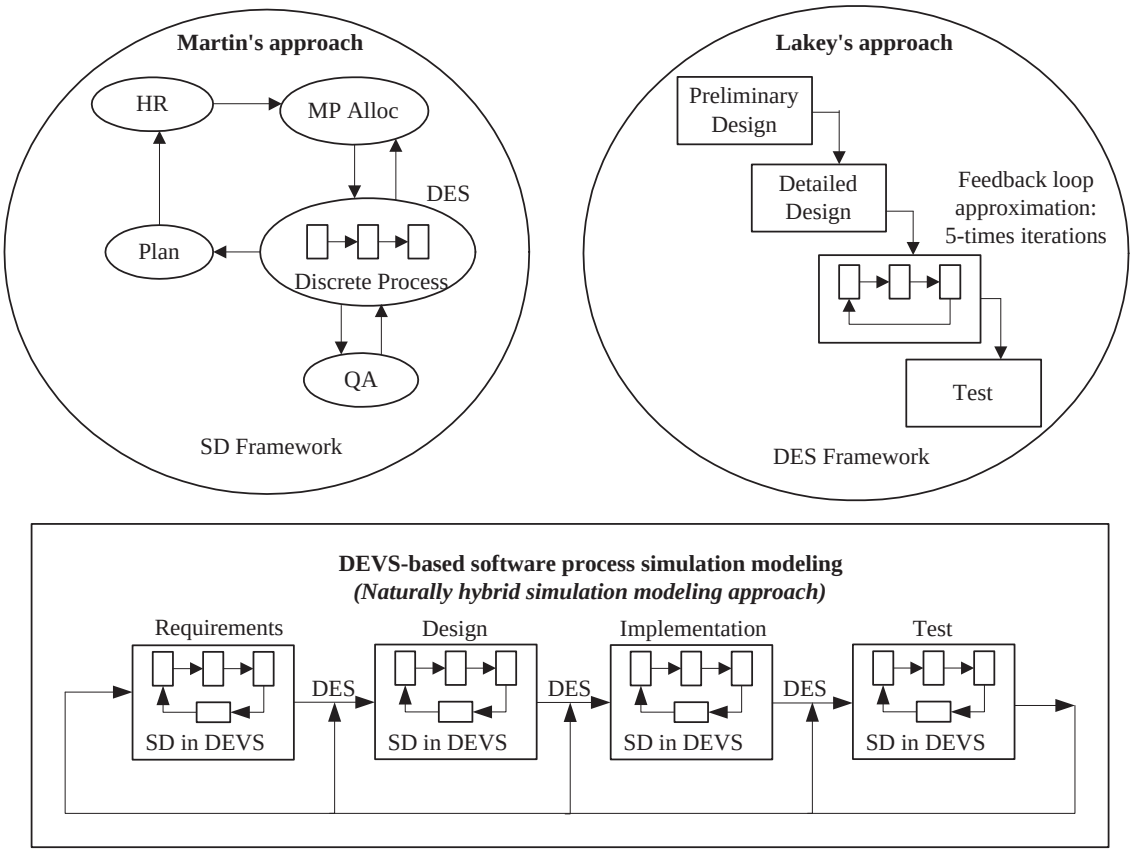


Figure 9. Characteristics of each hybrid simulation approach

namics. Within each of the sub-blocks are a number of equations that relate the product, process, and project factors to output parameters: Schedule, size, quality, manpower, overhead, skill level, tool support, process maturity, and functional growth. The limitation of this approach is that this may not fully represent overall dynamic feedback relationships in software development project because it approximates feedback loops by limited five iterations in each process activity.

Martin et al. [11] advance the clock at a small steady increment and integrate all necessary equations to implement system dynamics concept, and Lakey [12] iterates Development, Review, and Rework sub-activities five times in each activity to incorporate dynamic feedback loops. Martin combines discrete event model into the system dynamics framework, and Lakey approximates feedback loop in discrete event model. On the other hand, our approach embeds DESS and DTSS into DEVS formalism, and implements each activity phase (e.g., Requirements) with system dynamics model with DEVS as shown in Figure 9. This simulation modeling environment naturally includes both discrete and continuous components.

5. Conclusion and Future Work

We proposed DEVS-based software process simulation modeling technique which is a formally specified, modularized, and extensible simulation modeling approach. We modularized the simulation model by encapsulating closely related variables in one atomic model, and used four methods to specify a software process simulation model. Our approach enables us to make a clear, verifiable, understandable, and extensible specification for a software process simulation model. DEVS-based software process simulation modeling technique also overcomes some limitations of existing system dynamics models such difficulties as describing discrete process steps, controlling the activity sequences, and modeling uncertainties, by providing naturally hybrid simulation modeling environment.

This simulation modeling approach is unique in that it adopts DEVS formalism, a general purpose modeling and simulation framework, to a software process simulation modeling domain and implements a naturally hybrid simulation environment by embedding DESS and DTSS into DEVS formalism. We can implement the same system dynamics models as models developed by system dynamics

modeling tools (Vensim, Extend, or iThink) with discrete event simulation technique, and at the same time we can utilize the advantages of discrete event simulation because this technique is fundamentally a discrete event simulation one.

We have described a base model using DEVS-based software process simulation modeling technique and shown some examples of extended model. We, however, haven't experimented in industrial environment. We have plan to experiment a Waterfall life-cycle model in industrial setting, which will makes this simulation modeling approach more concrete.

References

[1] D. Raffo, G. Spehar, and U. Nayak, "Generalized Simulation Models: What, Why and How?", *ProSim '03*, 2003.

[2] N. Angkasaputra and D. Pfahl, "Making Software Process Simulation Modeling Agile and Pattern-based", *ProSim '04*, 2004.

[3] M. Ruiz, I. Ramos, and M. Toro, "A simplified model of software project dynamics", *The Journal of Systems and Software*, Elsevier, 2001, pp. 299-309

[4] T. Abdel-Hamid and S. Madnick, *Software Project Dynamics: An Integrated Approach*, Prentice-Hall, Englewood Cliffs, NJ, 1991

[5] Ling Liu, "EVOLVE: adaptive specification techniques for object-oriented software evolution", *Thirty-First Hawaii International Conference*, 1998

[6] B. Zeigler, H. Pracehofer, and TagGon Kim, *Theory of Modeling and Simulation, Second Edition*, Academic Press, New York, 2000

[7] TagGon Kim, *DEVSimHLA v2.2.0 Developer's Manual*, Korea Advanced Institute of Science and Technology (KAIST), 2004

[8] E. Kofman, M. Lapadula, and E. Pagliero, "PowerDEVS:A DEVS-Based Environment for Hybrid System Modeling and Simulation, *Technical Report LSD0306*, LSD, Universidad Nacional de Rosario, 2003

[9] *Vensim Modeling Guide*, Ventana Systems, Inc., 2004

[10] B. Boehm, *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, NJ, 1981

[11] R.H. Martin and D. Raffo, "A Model of the Software Development Process Using Both Continuous and Discrete Models", *Software Process Improvement And Practice*, John Wiley & Sons, NJ, 2000, pp. 147-157

[12] Peter B. Lakey, "A Hybrid Software Process Simulation Model for Project Management", *ProSim '03*, 2003.

Towards an Agile Development Method of Software Process Simulation

Niniek Angkasaputra, Dietmar Pfahl
Fraunhofer Institute for Experimental Software Engineering
Kaiserslautern, Germany
{angkasa, pfahl}@iese.fraunhofer.de

Abstract

Process simulation modeling has been used for various purposes in software engineering. To become even more widely used, in particular in industrial environments, an effective and efficient modeling method is required. IMMoS (Integrated Measurement, Modeling, and Simulation) offers such a method. Once a valid software process simulation model has been developed, it is important to maintain the model, i.e., to keep it up-to-date with the current software development process. This requires a simulation modeling process that shortens delivery time and is responsive to changes of the modeled process. One way to achieve this is to make simulation modeling agile. This paper describes our work on enhancing IMMoS to make it an agile method, and by adopting the practices of a suitable agile software development method.

1 BACKGROUND AND MOTIVATION

To be successful, companies try to satisfy their customers by delivering quality software on time and within budget. As software development happens in very dynamic environments, flexibility of the development processes is an important success factor in this endeavor.

During the last two decades, process simulation modeling has been used for various purposes in software engineering, i.e., strategic management, planning, control and operational management, understanding, and process improvement [9]. Model validity is an important success criterion of all software process simulation (SPS) projects. To be cost-effective, a valid SPS model must not only be able to adequately reflect the modeled system at a particular point in time, but over long time periods. This implies that an SPS model has to be kept up-to-date with the current software process, i.e., the modeled system. Taking into account that software processes may be instable, the SPS modeling process must be able to produce and re-adjust SPS models in a short delivery time. A study in the early 1990s found that development of SPS models is very time-consuming [8]. Unfortunately, this is still the case to date. In order to overcome this problem, we started to investigate possibilities to make SPS modeling processes more flexible, and thus more efficient. To tackle this goal, we followed a three-step approach.

First, we selected an existing SPS modeling method that provides sufficiently mature process guidance. As a point of reference, we focused on an existing, well-documented and mature SPS modeling process offered by the IMMoS (Integrated Measurement, Modeling, and Simulation) framework [10, 11]. Although IMMoS focuses on the development of continuous SPS models, i.e., System Dynamics (SD) models, most of the elements of IMMoS can also be used for the development of other types of SPS models, e.g., discrete-event models.

Second, we identified agile software development as one source of inspiration for making IMMoS more flexible and thus more efficient and cost-effective [4]. We reviewed existing survey reports to understand the principles and characteristics of agile methods known in the area of software engineering (SE). Inspired by Abrahamsson's characterization scheme of agile software development methods [1], we analyzed to what extent IMMoS implicitly exhibits corresponding characteristics. Based on this analysis we re-formulated and slightly enhanced IMMoS to make its agility more explicit and complete. To reflect these changes we assigned the name Agile-IMMoS to the enhanced SPS modeling method.

Third, we chose one agile method that implements most of the common agile principles, i.e., Extreme Programming, and used it as a model for proposing further enhancements of Agile-IMMoS through adoption of suitable agile practices which are not yet considered in the various activities of the SPS modeling process.

The remainder of this paper is organized as follows. Section 2 briefly presents the SPS modeling guidance offered by IMMoS and discusses to which extent agility is implicitly present. Section 3 gives an overview of existing agile methods, summarizes the commonalities of agile development processes, and lists their practices. Section 4 presents Agile-IMMoS, the result of enhancing IMMoS towards an agile SPS modeling method. Section 5 summarizes existing related work. Section 6 presents conclusions and plans for future work.

2 THE IMMoS METHODOLOGY

The following problems motivated the development of the IMMoS methodology [11]:

- Lack of a commonly accepted and sufficiently detailed SPS modeling process;
- Lack of precise guidelines on how to define the SPS problem statement in a goal-oriented way;
- Lack of reuse of information from software engineering (SE) models that typically exist in software organizations, i.e., static quantitative models, and static process models;
- Lack of integration of SPS modeling with established static modeling methods to create synergy effects.

IMMoS addresses all of these problems. In this section, the IMMoS methodology is briefly described to give an overview[1].

2.1 The IMMoS Framework

The framework of IMMoS process guidance for SD-based SPS modeling consists of the following models:

IMMoS Phase Model. Four phases are defined to structure the SDM[2] lifecycle: Phase 0 – pre-study to identify SDM users and define the SDM modeling goal, Phase 1 – development of an initial SDM that reproduces the system's reference behavior, Phase 2 – enhancement of the initial SDM to make it suitable for the intended problem analysis/solution, and Phase 3 – application and maintenance of the enhanced SDM. It is important to note that these phases can be iterated.

IMMoS Role Model. This model defines the minimal set of roles that are important in an SDM development project: SDM Customer (C) as the sponsor of the SDM development project, SDM User (U) as the future user of the SDM in the customer's organization, SDM Developer (D) as the responsible person for the technical development, Facilitator (F) as a support for establishing contacts and planning as well as arranging meetings, Moderator (M) for preparing and guiding workshops and meetings, and SE Subject Matter Expert (E) as source of relevant SE information needed for SDM building.

IMMoS Product Model. This model provides a complete list of work and end products in the different phases. Among the products defined, several occur in any SDM modeling process: SDM Dynamic Hypotheses Definition (i.e., Reference Mode and Base Mechanisms), Causal Diagram, and SDM (i.e., Initial Flow Graph and Mathematical Equations). IMMoS distinguishes three SDM maturity levels: the Initial SDM of phase 1, and the Enhanced SDMs of phases 2 and 3.

IMMoS Process Model. The IMMoS process model defines the activities that are (repeatedly) performed in the various phases. Each activity is described by eleven attributes: activity ID, name, (involved) role, input, output, entry condition, exit condition, description (of tasks to be performed), (useful) methods and techniques, (special) guidelines, and (provided) materials and tools. The IMMoS process activities are associated to the SDM lifecycle as follows:

- Phase 0 (pre-study) consists of the following activities: 1) first contact, 2) characterization, 3) management briefing, 4) identification of SDM users, 5) problem definition, 6) technical feasibility check, and 7) planning and contract.
- Phase 1 (initial model development) consists of the following activities: 1) technical briefing, 2) definition of dynamic hypotheses, 3) definition of causal diagram, 4) review of the causal diagram (verification 1), 5) implementation of initial SDM, 6) review of the initial SDM (verification 2), and 7) test of the initial SDM (validation 1).
- Phase 2 (model enhancement) consists of the following activities: enhancement of the initial SDM and test of the enhanced SDM (validation 2). Note that the activity enhancement of the initial SDM basically repeats the activities in Phase 1.
- Phase 3 (model application and maintenance) consists of the activity application and maintenance of the SDM. The user applies the SDM to find new policies that solve the problem(s) at hand. This includes experimenting with parameter values and model structures. If modification of the SDM is necessary, then activities similar to Phase 2 are repeated.

2.2 The Agility of IMMoS

Based on our experience from many SPS modeling projects, we found that the following five requirements should be fulfilled by SPS modeling methods:

R1: Involvement of domain experts is required in the definition of a valid model that is close to the real system.

R2: The validation task is iterated until a model is considered to be satisfying the system's reference mode.

R3: Dynamic user requirements should be responded to quickly and with the right details.

R4: The development process should better be performed top-down, i.e., starting from a very abstract and simple model to a more detailed and complex model.

R5: Involvement of a variety of roles is required, not only from the simulation model developers but also from the customer's management.

[1] A more detailed summary can be found in [11]. The complete work has been published in [10].

[2] SDM is an abbreviation for "SD model".

IMMoS was developed based on the fundamental assumption that continuous alteration of an SDM should be expected and managed in order to keep it consistent with the change-prone software processes that it intends to represent. Due to this underlying assumption, IMMoS fulfils these requirements to a high degree.

The IMMoS methodology fosters communications between SDM developers, SE experts and key stakeholders of the customer's organization (R1 and R5). This is realized through the definition of different roles in the IMMoS role model and their involvement in the activities defined by the IMMoS process model.

The IMMoS process model defines activities that are iterated until a valid SDM is delivered. It also defines review activities that do not only aim at finding faults in work products early (R2). The reviews also give the opportunity to capture change requests (R3).

Furthermore, the IMMoS phase model distinguishes several development stages that SDMs undergo during development, from initial/simple models to enhanced/complex models (R4).

In software engineering, the following agile principles are known for providing better support for the customer and timely delivery of the desired software [2]:

- P1: Individuals and interactions over processes and tools
- P2: Working software over comprehensive documentation
- P3: Customer collaboration over contract negotiation
- P4: Responding to change over following a plan

In this list, the word "over" stands for "is/are at least as important as". It implies that the expressions on the left hand side and on the right hand side of the word "over" should be kept in balance. The four principles have been implemented in various agile software development methods and several positive effects have been reported [1].

The degree to which IMMoS addresses the requirements R1 to R5 indicates that it implicitly supports to some extent the agile principles listed above. First, IMMoS keeps processes, tools, and contract negotiation balanced with communication and collaboration between the SDM developer and the partners in the customer organization (P3). Second, IMMoS balances comprehensive documentation with demonstrating valid work and end products to the customer during review (P2). Third, IMMoS balances adherence to the project plan with responsiveness to change requests from the customer (P4).

Nevertheless, IMMoS cannot yet be considered a fully practical agile method for SPS model development. For example, agile principle P1 is not sufficiently well implemented. Currently, IMMoS process guidance for SDM development is documented in a rather monolithic way, giving the (not intended) impression that processes and tools are more important than the interaction between the individuals that assume the various roles described in the method. Thus, additional work is required to enhance IMMoS and to make the agility of IMMoS more explicit and practical. This will be done by re-formulating the modeling guidance of IMMoS to make it more compact, flexible, transparent, and supportive towards agility.

3 AN OVERVIEW OF AGILE METHODS

In this section, we first summarize common characteristics of agile (software development) processes. Then we present existing agile methods in SE, and list the practices that they entail, indicating which of the four agile principles each practice mainly focuses on.

3.1 Characteristics of Agile Processes

Through a process of deduction from various discussions of other authors (some of them authors of agile methods) about the characteristics of agile software processes, Abrahamsson et al. [1] summarized the following important characteristics that make a software development method agile:

- Incremental (C1): Development happens in small releases with rapid development cycles in order to deliver executable programs fast and to get immediate feedback for enhancement.
- Cooperative (C2): Development stresses the importance of customer and developers working constantly together with close communication.
- Adaptive (C3): Development is able to react to last moment changes.
- Straightforward (C4): The method itself is easy to learn and to modify and it is sufficiently documented.

In Section 4, we will use characteristics C1 to C4 as our main points of reference for the transformation of IMMoS into Agile-IMMoS.

3.2 Agile Methods and their Practices

In our attempt to find an agile method that could serve as a reference for enhancing Agile-IMMoS towards more agility, we identified ten candidate methods from four main sources: Seven agile methods described in [1], one (i.e., Agile Software Process) in [5], one (i.e., Agile Modeling) in [3], and another one (i.e., Lean Software Development) in [12]. We then associated the practices of each agile method to one of the agile principles they mainly focus on (see Table 1).

We briefly summarize the essentials of the agile methods listed in Table 1 in the next paragraphs.

Table 1. Agile methods with their practices clustered with regards to agile principles

	Agile Principles			
Agile Methods	**P1: Individuals & interactions**	**P2: Working software**	**P3: Customer collaboration**	**P4: Responding to change**
Dynamic Systems Development Method (DSDM)	• DSDM teams must be empowered to make decisions • A collaborative and cooperative approach shared by all stakeholders is essential	• Frequent delivery of product • Iterative and incremental development • Changes are reversible • Requirements are baselined at a high level • Testing is integrated throughout the lifecycle	• Active user involvement is imperative • Fitness for business purpose • A collaborative and cooperative approach shared by all stakeholders is essential	
Scrum	• Product Backlog • Daily Scrum meeting	• Sprint planning meeting • Effort estimation • Sprint Backlog	• Sprint planning meeting • Sprint review meeting	• Sprint procedure • Sprint review meeting
Crystal Family	• Holistic diversity strategy • Reflection workshops	• Staging • Revision and review • Monitoring • Parallelism and flux • Methodology-tuning technique	• User Viewings	
Extreme Programming (XP)	• Metaphor • Pair programming • Collective code ownership • Coding standards	• Small releases • Simple design • Continuous integration • Continuous testing • 40-hour week	• Planning game • On-site Customer	• Refactoring
Adaptive Software Development (ASD)		• Iterative development • Feature-based planning	• Customer focus group reviews	
Pragmatic Programming (PP)		• Rigorous testing • Incremental & iterative development	• User-centered design	
Agile Software Process (ASP)	• People-centered process	• Time-based process • Concurrent and asynchronous process • Incremental delivery and iterative process • Distributed software process		
Feature-driven Development (FDD)	• Individual Class (Code) Ownership • Feature teams • Progress Reporting	• Domain Object Modeling • Developing by Feature • Inspection • Regular builds • Configuration management		
Agile Modeling (AM)	• Teamwork, e.g., model with others, collective ownership • Motivation: e.g., model to communicate • Productivity, e.g., apply modeling standards, reuse existing resources	• Iterative and incremental modeling, e.g., apply the right artifacts, create several models in parallel • Simplicity, e.g., create simple content • Validation, e.g., consider testability • Documentation, e.g., discard temporary models	• Teamwork, e.g., active stakeholder participation	
Lean Software Development (LSD)	• Amplify learning • Empower the team • See the whole	• Eliminate waste • Deliver fast	• Build integrity in	

The *Dynamic Systems Development Method (DSDM)* suggests fixing time and resources and then adjusting the amount of functionality accordingly instead of fixing the amount of functionality in a product and then adjusting time and resources accordingly. The items listed in the table are more in the form of DSDM principles than practices.

Scrum is an approach for managing the systems development process in a constantly changing development environment.

Crystal Family consists of a set of different methods to be selected and provides suggestions on choosing an appropriate methodology for a project based on its size and criticality. The method is open for any development practices and tools.

Extreme Programming (XP) has the novelty of being based on the way the individual practices are collected and lined up to function with each other.

Adaptive Software Development (ASD) offers solutions for the development of large and complex systems, in particular. The method encourages incremental and iterative development with constant prototyping.

Pragmatic Programming (PP) introduces a set of programming practices. The PP method itself comprises a collection of short programming tips that focus on day-to-day problems.

The *Agile Software Process (ASP)* method focuses on delivering products incrementally over time. The ASP is enacted iteratively with a fixed cycle-time.

The *Feature-driven Development (FDD)* method focuses on the design and building phases. FDD consists of a set of best practices, emphasizes quality aspects throughout the process, and includes frequent and tangible deliveries along with accurate monitoring of project progress.

Agile Modeling (AM) applies the idea of agile, rapid development to software system modeling. Its key focus is on modeling practices and cultural principles. The method encourages developers to produce sufficiently advanced models to support acute design needs and documentation purposes. The items listed in the table are categorical practices, each of which contains several AM practices (given as examples).

Lean Software Development (LSD) aims at utilizing the same high level principles as guidelines or thinking tools for increasing software production efficiency. The items listed in the table are LSD's principles, which are guideposts for devising appropriate practices for a particular environment. LSD provides various tools for converting these principles into agile software development practices.

Based on the information contained in Table 1, we selected Extreme Programming (XP) created by Beck [6] as the candidate for improving Agile-IMMoS. This was due to the fact that XP seems to cover the agile principles most comprehensively. Moreover, XP is a lightweight methodology for small to medium sized teams developing software in the face of vague or rapidly changing requirements. This perfectly matches the context of SPS modeling. Also, XP is the most documented one of the various agile methods, and many successful applications have been reported [1]. In Section 4, we discuss to which extent relevant XP practices are yet reflected by Agile-IMMoS, and we identify simulation modeling activities that can be further improved by adopting XP practices not yet comprised in Agile-IMMoS.

4 AGILE-IMMoS

The enhancement of IMMoS, which aims at bringing its agile potentials to the surface, yielded Agile-IMMoS. Major modifications relate to the IMMoS process, phase, and product models. The IMMoS role model remains unchanged, but the mapping of roles to activities has been re-adjusted and presented more compactly.

Compared to traditional IMMoS, in Agile-IMMoS the characteristics of agile processes (cf. Section 3.1) are addressed more explicitly:

- Due to modifications to the IMMoS process and phase models, Agile-IMMoS becomes a more straightforward (C4) and adaptive (C3) SPS modeling methodology.
- Due to enhancements of the IMMoS product model, Agile IMMoS becomes a more incremental (C1) SPS modeling methodology.
- Finally, a more comprehensive description of how the existing IMMoS roles interact during SPS model development makes Agile-IMMoS a more cooperative (C2) SPS modeling methodology.

More detailed presentations of the models contained in Agile-IMMoS give Sections 4.1 to 4.4 below.

In order to further improve Agile-IMMoS with regards to agility, we analyzed the practices advocated by the agile SE method XP with regards to their relevance for inclusion in Agile-IMMoS. We then check whether relevant practices are already reflected in Agile-IMMoS. For relevant practices that are not yet reflected we describe how they should be included into Agile-IMMoS (cf. Section 4.5).

4.1 Agile-IMMoS Process Model

In Agile-IMMoS, the following modifications to the existing process model were made:

- The overview presentation of the IMMoS process model was altered and simplified.
- The activities in IMMoS Phase 0 were now capsulated as organizational activities, since they concern organizational aspects such as project orientation and planning until contract preparation.

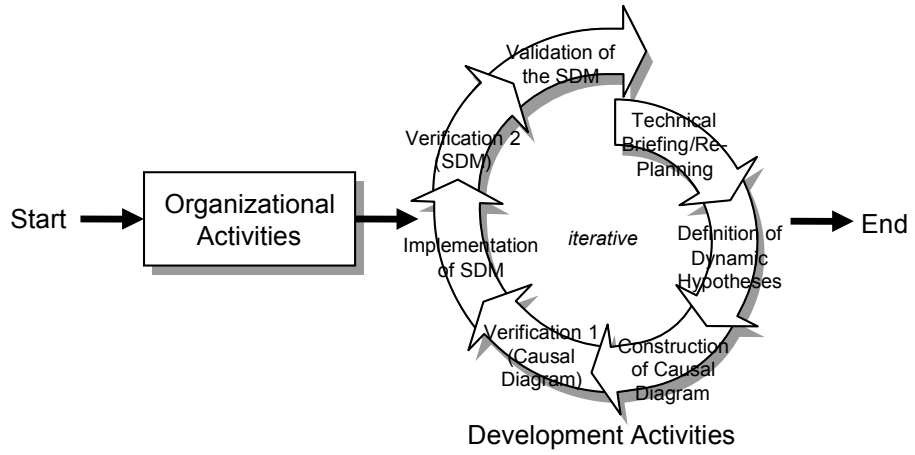


Figure 1. Process model overview of Agile-IMMoS

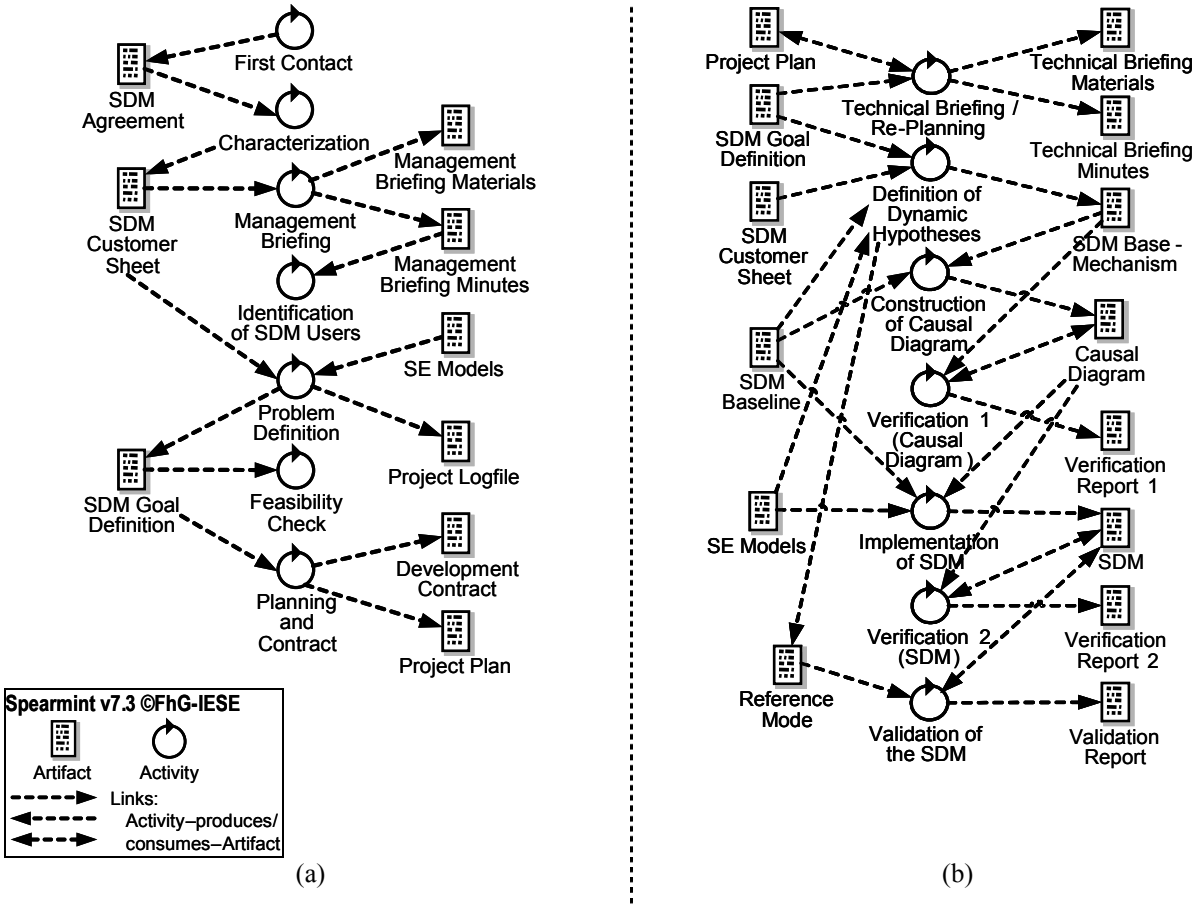


Figure 2. Process model of Agile-IMMoS: organizational activities (a) and development activities (b)

- The activities in IMMoS Phases 1 were generalized and packaged into a set of development activities that are iterated for developing and maintaining defined SDM releases. As a consequence, the activities in IMMoS Phases 1, 2 and 3 were collapsed.

In this way, the activities are no longer tied directly to the phase model (i.e., SDM lifecycle). The new overview presentation of the Agile-IMMoS process model is shown in Figure 1. The process starts with organizational activities followed by the SDM development activities, which are iterative. The amount of iterations and the length of any iteration depend on the SDM release plan.

The Agile-IMMoS process model now supports an iterative development process more explicitly with short cycles and clearly defined milestones. Short development cycles facilitate early risk and problem identification and enable rapid correction and

verification. Explicit milestones help to better plan and review development progress. Moreover the number of different types of SDM product has been reduced.

4.1.1 Organizational Activities

Figure 2 (a) presents the product flow of the organizational activities in Agile-IMMoS, which consists of seven activities. The descriptions of these activities are identical to those in the traditional IMMoS process model, except that the activity 'Planning and Contract' now comprises an additional task related to the planning the SPS model release. A typical product release plan is described in Section 4.3. For one SDM development project, the overall organizational activities are normally required to be performed only once.

4.1.2 Development Activities

Figure 2 (b) shows the product flow of the development activities in Agile-IMMoS. Again, it consists of seven activities: Technical Briefing / Re-Planning, three activities for constructing the main SDM products (Definition of Dynamic Hypotheses, Construction of Causal Diagram, and Implementation of SDM), and three V&V activities (Verification 1: Causal Diagram, Verification 2: SDM, and Validation of the SDM).

The number of activities is kept small to obtain a short development cycle. The milestones, which are used to plan and review development progress, are "start", "verification 1", "verification 2", "validation", and "finish/delivery".

Typically, development activities are iterated several times in any phase of the SDM lifecycle (cf. Section 4.2), and apply to any SDM release (cf. Section 4.3).

4.2 Agile-IMMoS Phase Model

Different from the traditional IMMoS methodology, SDM lifecycle phases in Agile-IMMoS are not directly associated with process activities (cf. Section 4.1). Instead, the Agile-IMMoS phase model structures the SDM lifecycle into three phases, which are linked to the SDM goal definition. An SDM goal consists of five elements: scope, purpose, role, dynamic focus, and environment. Using the set of SPS modeling purposes proposed in [9], Agile-IMMoS phases can be linked to SDM goal definitions as follows:

- Phase 1: Initial SDM, to reproduce the reference mode. A reference mode is an explicit description of the (problematic) dynamic behavior of one or more system parameters observed in reality.
- Phase 2: Enhanced SDM for singular problem solving such as strategic management as well as process improvement and technology adoption.
- Phase 3: Enhanced SDM for any repeated use, for the same purpose such as management planning, control and operational management, understanding, as well as training and learning. This phase likely involves maintenance of the model to keep it valid compared to the current real system.

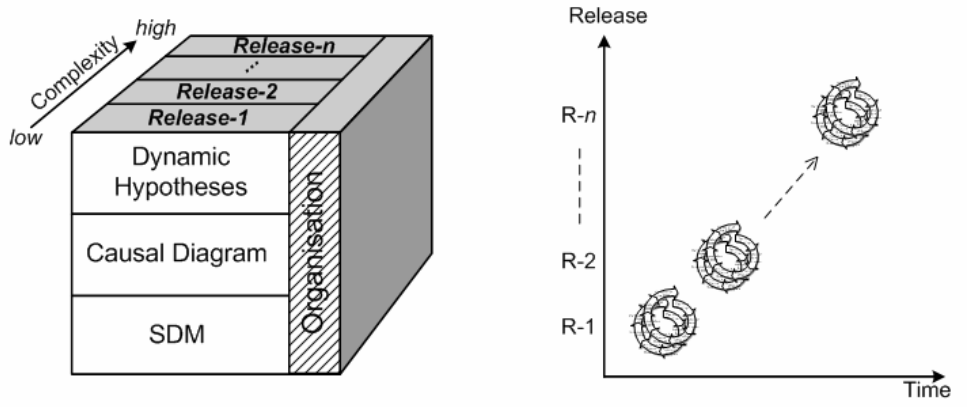
4.3 Agile-IMMoS Product Model

Several intermediate and end products produced by Agile-IMMoS activities are shown in Figure 2. The Agile-IMMoS product model defines these products and provides templates to support the developers. Among the products, a core set of products that are typical for any SDM modeling project are listed below:

a. Base Mechanism and Reference Mode, both constituting the Dynamic Hypotheses of the SDM;
b. Causal Diagram;
c. SDM, consisting of the SDM Flow Graph and its underlying Mathematical Equations, as well as (optionally) a Graphical User Interface.

These products are developed in small releases to get an incremental development that makes it possible to deliver a meaningful model fast and to get immediate feedback for correction and enhancement. Each phase can have several releases. Often, releases are determined based on the complexity of the SDM (i.e., from low to high complexity) as illustrated in Figure 3 (a), but there can also be other considerations, e.g., constraints in availability of experts or data regarding certain aspects of the problem to be resolved. SDM releases are initially planned during the activity 'Planning and Contract' of organizational activities. During SDM development, re-planning can be done during activity 'Technical Briefing / Re-Planning'.

For any product release, iterations of development activities are planned and performed as illustrated in Figure 3 (b). Ideally, an iteration will not take longer than two calendar weeks. The exact number of iterations varies depending on the novelty and/or difficulty of the problem, the number of proposed solution alternatives, the experience and skill levels of the developers with the simulation approach, and the availability of subject matter experts and prospective users from the customer organization.



(a) (b)

Figure 3. SDM release planning

4.4 Agile-IMMoS Role Model

The role types and their descriptions do not change from the IMMoS role model (cf. Section 2.1 for the list of roles); only a new summary representation is added to the Agile-IMMoS method to give a sense of the degree of involvement to the persons assuming the various roles in a modeling project.

Table 2

(a) Role involvement in organizational activities

	C	U	D	F	M	E
O1: First Contact	✓		✓			
O2: Characterization	✓		✓	✓		
O3: Management Briefing	✓	✓	✓	✓		
O4: Identification of SDM Users	✓	✓				
O5: Problem Definition	✓	✓	✓			
O6: Feasibility Check	✓	✓	✓	✓		✓
O7: Planning and Contract	✓		✓			

(b) Role involvement in development activities

	C	U	D	F	M	E
D1: Technical Briefing / Re-Planning	✓	✓	✓	✓		✓
D2: Definition of Dynamic Hypotheses		✓	✓	✓	✓	✓
D3: Construction of Causal Diagram			✓			
D4: Verification 1 (Causal Diagram)		✓	✓	✓	✓	✓
D5: Implementation of SDM		✓	✓	✓		✓
D6: Verification 2 (SDM)			✓			
D7: Validation of the SDM		✓	✓	✓	✓	✓

Although involvement of people from the customer organization is important, it should not be excessive in order to avoid being counter-productive. People are involved only to some degree, which is expressed in three shades of color intensity: the darker the cell shadow, the more intensively the role is involved in terms of effort. Table 2 (a) and (b) shows the role involvement in the Agile-IMMoS organizational activities (O1-O7) and development activities (D1-D7), respectively.

4.5 Agile Practices in Agile-IMMoS

In order to further improve Agile-IMMoS with regards to agility, we first analyzed the twelve practices advocated by the agile SE method XP with regards to their relevance for inclusion in Agile-IMMoS. Then we checked whether relevant practices are yet implicitly or explicitly reflected in Agile-IMMoS. The results are presented in Table 3.

Table 3. Coverage of XP practices in Agile-IMMoS

XP Practice	Agile-IMMoS Framework	
	Relevant	**Reflected**
Metaphor	yes	yes
Pair programming	yes	no
Collective code ownership	yes	yes
Coding standards	yes	yes
Small releases	yes	yes
Simple design	yes	yes
Continuous integration	yes	yes
Continuous testing	yes	yes
40-hour week	yes	no
Planning game	yes	no
On-site customer	yes	yes
Refactoring	yes	yes

It turned out that all practices are relevant. Nine of them are already reflected in the Agile-IMMoS framework. Three practices are relevant but not yet reflected (light gray shaded cells).

'Metaphor' is implicitly reflected through the Agile-IMMoS product 'Dynamics Hpotheses Definition (containing 'Base Mechanisms' and 'Reference Mode'). 'Collective code ownership' is the standard practice in SDM development. 'Coding standards' are supported by the SD modeling tools. 'Small releases' are supported by the Agile-IMMoS phase and product models, and the related guidelines in the process model. 'Simple design' is supported by the guidelines for the definition of base mechanisms provided by the development activity 'Definition of

Dynamic Hypotheses'. 'Continuous integration' is facilitated through the functionality of standard SD development tools which allows for developing and executing sub-models either separately or combined. 'Continuous testing' is facilitated through the early availability of reference modes for validation testing. 'On-site customer' is supported by the Agile-IMMoS role and process models. 'Refactoring' is addressed by the guidelines for developing 'Causal Diagrams' in the development activity 'Construction of Causal Diagram'.

Table 4 shows in which activities of the Agile-IMMoS process model the four agile practices that are not yet properly reflected, should be included (possibly after adaptation).

Table 4. XP practices to be introduced into Agile-IMMoS

XP Practice	Agile-IMMoS Activities	
	Organizational	**Development**
Pair Programming		D2, D3, D5
40-hour week	O7	D1
Planning Game	O7	D1

'Pair programming', one of the most popular XP practices, advocates the idea of having two programmers working together using a single computer and giving each other immediate feedback. This practice could easily be encouraged by explicitly including it as a recommended technique for all roles involved in development activities D2, D3, and D5 (Definition of Dynamic Hypotheses, Construction of Causal Diagram, Implementation of SDM). The technique could not only apply to the role SDM Developer, but expand to roles SDM User and SE Subject Matter Expert.

'40-hour week' advocates a strict risk management with regards to time and effort consumption. This can be reflected in Agile-IMMoS activites O7 and D1 (Planning and Contract, Technical Briefing / Re-Planning) by adding the 40-hour week rule to the guidelines applying to the re-/planning tasks.

'Planning game' advocates the idea of bringing together customers, subject matter experts, and modeler for the purpose of defining releases that provide the maximal business value. Again, this can be added as a specific technique to Agile-IMMoS activities O7 and D1.

5 RELATED WORK

By making IMMoS more agile, we tried to provide a comprehensive SPS modeling methodology that helps increase the acceptance of process simulation in software industry by making it more efficient and cost-effective. In the literature, we have not found other proposals aiming at the same target that are equally comprehensive. However, several useful efforts have been made to provide SPS modeling techniques that make certain steps of SPS modeling more efficient and cost-effective. These techniques include templates [7], generic models [13], pattern-based generators [14], and building blocks [15]. All of these techniques could be adapted and integrated into Agile-IMMoS process activities.

6 CONCLUSION AND FUTURE WORK

Agile-IMMoS offers an agile development approach to SPS modeling with System Dynamics. The Agile-IMMoS modeling guidance consists of a process model, phase model, product model (incl. release concept), and role model, which are arranged in such a way that agile values are put upfront. In particular, this arrangement enables flexibility in defining and adjusting project plans and model scopes in the form of SDM releases.

Based on our analyses, we conclude that Agile-IMMoS exhibits all characteristics of an agile method, and supports all agile principles. Moreover, when taking XP as a reference method from the area of software engineering, we found that nine out of twelve practices are reflected by Agile-IMMoS. The three remaining practices can easily be added to the method.

Furthermore, Agile-IMMoS provides certain potential practices that can be applied to the development activities. These practices were adapted from the agile method Extreme Programming, in addition to the methods and techniques, special guidelines, as well as materials and tools that were already in IMMoS.

In the current version, Agile-IMMoS covers all important SPS modeling activities from project definition (i.e., organizational activities) to SDM development (i.e., development activities). It does not yet address issues related to the training of SPS model users.

Our future work will focus on the evaluation of Agile-IMMoS with regards to efficiency and cost-effectiveness in SPS modeling project using System dynamics. We also intend to adapt and formalize Agile-IMMoS to support the development of discrete-event SPS models.

7 ACKNOWLEDGEMENT

The authors would like to thank Sonnhild Namingha from the Fraunhofer Institute for Experimental Software Engineering (IESE) for the editorial review of the first version of this article.

8 REFERENCES

[1] Abrahamsson, Pekka; Salo, Outi; Ronkainen, Jussi; Warsta, Juhani: "Agile software development methods: Review and Analysis". Technical Research Centre of Finland, VTT Publications 478, available online:

http://www.inf.vtt.fffpdf/publications/2002/P478.pdf, Espoo, Finland, 2002.

[2] Agile Alliance: “Agile Manifesto”, source: official Website http://www.agilemanifesto.org.

[3] Ambler, Scott W.: Agile Modeling: Effective Practices for Extreme Programming and the Unified Process, John Wiley & Sons, Inc. , New York, 2002.

[4] Angkasaputra, Niniek; Pfahl, Dietmar: “Making Software Process Simulation Modeling Agile and Pattern-based”. In: Proceedings of the 5th Process Simulation Modelling Workshop (ProSim-2004), Edinburgh, Scotland, 24-25 May 2004, pp. 222-227.

[5] Aoyama, Mikio: “Agile Software Process and Its Experience”. In: Proceedings of the 20th International Conference on Software Engineering (ICSE), Kyoto, Japan, IEEE Computer Society, Washington, DC, USA, 1998, pp. 3-12.

[6] Beck, Kent: Extreme Programming Explained: Embrace Change, Addison-Wesley, 2000.

[7] Berling, Tomas; Andersson, Carina; Höst, Martin; Nyberg, Christian: "Adaptation of a Simulation Model Template for Testing to an Industrial Project". In: Proceedings of the 4th Process Simulation Modelling Workshop (ProSim-2003), Portland, USA, May 2003.

[8] Keller, Lucien; Harrell, Charles; Leavy, Jeff: “The Three Reasons Why Simulation Fails”. In: Industrial Engineering, Vol. 23, No. 4, IE, April 1991, pp. 27-31.

[9] Kellner, Marc I.; Madachy, Raymond J.; Raffo, David M.: “Software process simulation modelling: Why? What? How?” In: The Journal of Systems and Software, Vol. 46, Elsevier, USA, 1999.

[10] Pfahl, Dietmar: An Integrated Approach to Simulation-Based Learning in Support of Strategic and Project Management in Software Organisations. PhD Theses in Experimental Software Engineering, Fraunhofer IRB Verlag, Stuttgart, 2001.

[11] Pfahl, Dietmar, Ruhe, Guenther: “IMMoS: A Methodology for Integrated Measurement, Modelling, and Simulation”. In: International Journal of Software Process: Improvement and Practice, Wiley, Vol. 7, 2003, pp. 189-210.

[12] Poppendieck, Mary: “Lean Software Development”. In: C++ Magazine, Fall 2003.

[13] Raffo, D.; Spehar, G.; Nayak, U.: "Generalized Simulation Models: What, Why and How?". In: Proceedings of the 4th Process Simulation Modelling Workshop (ProSim-2003), Portland, USA, 3-4 May 2003.

[14] Schuetze, M.; Riegel, J.P.; Zimmermann, G.: “A Pattern-Based Application Generator for Building Simulation”. In: Proceedings of the 6th European Conference held jointly with the 5th ACM SIGSOFT international symposium on Foundations of Software Engineering, Zurich, Switzerland, 1997, pp. 468-482.

[15] Valentin, E.; Verbraeck, A.: “Simulation Using Building Blocks”. In: Proceedings of AI, Simulation and Planning in Highly Automated Systems, 2002, pp. 65-71.

Section IV

Process Simulation Modeling – Focus on Model Implementation

Software Process and Business Value Modeling

Ray Madachy, *Senior Member, IEEE*

***Abstract* — Business value attainment should be a key consideration when designing software processes. Ideally they are structured to meet organizational business goals, but it is usually difficult to integrate the process and business perspectives quantitatively. This research uses modeling and simulation to assess tradeoffs for business case analysis. An improved model relates the dynamics between product specifications, investment costs, schedule, software quality practices, market size, license retention, pricing and revenue generation. The system dynamics model allows one to experiment with different product strategies, software processes, marketing practices and pricing schemes while tracking financial measures over time. It can be used to determine the appropriate balance of process activities to meet goals. Examples are developed for varying scope, reliability, delivery of multiple releases, and determining the quality sweet spot for different time horizons. Results show that optimal policies depend on various stakeholder value functions, opposing market factors and business constraints. Future model improvements are also identified.**

***Index Terms*—Software process modeling and simulation, Value-based software engineering, Software business value, Software quality, System dynamics**

I. Introduction and Background

Software-related decisions should not be extricated from business value concerns. Unfortunately, software engineering practice and research frequently lacks a value-oriented perspective. Value-Based Software Engineering (VBSE) integrates value considerations into current and emerging software engineering principles and practices [1]. This research addresses the planning and control aspect of VBSE to manage the value delivered to stakeholders. Techniques to model cost, schedule and quality are integrated with business case analysis to allow tradeoff studies in a commercial software development context. Business value is accounted for in terms of return-on-investment (ROI) of different product and process strategies.

It is a challenge to tradeoff different software attributes, particularly between different perspectives such as business and software development. Software process modeling and simulation can be used to reason about software value decisions. It can help find the right balance of activities that contribute to stakeholder value with other constraints such as cost, schedule or quality goals.

A value-oriented approach provides explicit guidance for making products useful to people by considering different people's utility functions or value propositions. The value propositions are used to determine relevant measures for given scenarios.

Two major aspects of stakeholder value are addressed here. One is the business value to the development organization stemming from software sales. Another is the value to the end-user stakeholder from varying feature sets and quality. Production functions relating different aspects of value to their costs were developed and are included in the integrated model.

A. Relation to Previous Work

An early version of the model in this research is described in [3]. Its focus was on the impact of quality on sales, market dynamics and financial measures with the software process largely stubbed out. A primary improvement has been the addition of a major sector in the model for software process and product. It covers the development process including effort, schedule and product quality in terms of defect levels.

Previously the staffing profile and expected quality were derived externally and manually input. Neither could be impacted dynamically during the course of a simulation run.

Now the software process and product sector is capable of producing the internal dynamics between job size, effort, schedule, required reliability and quality. Input parameters can also be modified interactively by the user during the course of a run and the model responds to the midstream changes. Additional investment options were also added to the finance sector. The model has been enhanced overall to be more general-purpose and user-interactive.

Also added to the model are value-based production functions relating the market value to the cost of different feature sets and to the cost of reliability. It is shown in the examples how these constructs can be used to help quantify the value of different strategies. The model can now be used to assess the effects of combined strategies by varying the scope and required reliability independently or simultaneously.

Ray Madachy is with the University of Southern California Center for Software Engineering, University of Southern California, Los Angeles, CA 90089 USA and Cost Xpert Group, San Diego, CA 92109 USA (e-mail: madachy@usc.edu).

II. Model Overview

The system dynamics model represents a business case for commercial software development. The user inputs and model factors can vary over the project duration as opposed to a static model. It can be used dynamically before or during a project. Hence it is suitable for "flight simulation" training or actual project usage to reflect actuals to-date.

The sectors of the model and their major interfaces are shown in Fig. 1. The software process and product sector computes the staffing profile and quality over time based on the software size, reliability setting, and other inputs. The staffing rate becomes one of the investment flows in the finances sector, while the actual quality is a primary factor in market and sales. The resulting sales are used in the finance sector to compute various financial measures.

Fig. 2 shows a diagram of the software process and product sector. It dynamically calculates effort, schedule and defects. The staffing rate over time is calculated with a version of Dynamic COCOMO [5] using a variant of a Rayleigh curve calibrated to the COCOMO II cost model at the top level. The project effort is based on the number of function points and the reliability setting. There are also some parameters that determine the shape of the staffing curve.

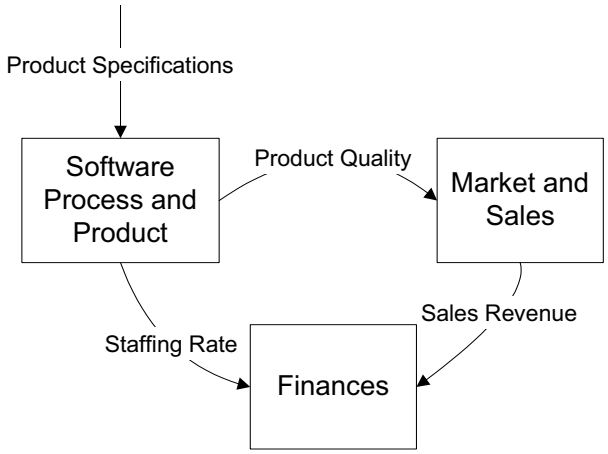


Fig. 1. Model sectors and major interfaces

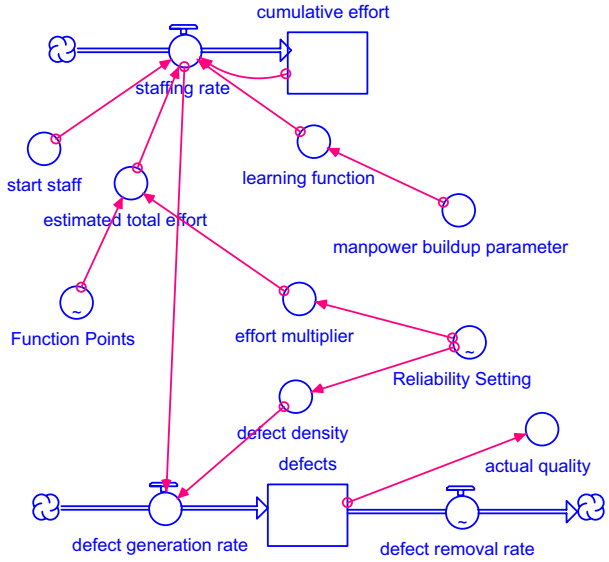


Fig. 2. Software process and product sector

There is a simple defect model to calculate defect levels used in the market and sales sector to modulate sales. Defect generation is modeled as a co-flow with the software development rate, and the defect removal rate accounts for their finding and fixing. See [4] for more background on these standard flow structures for effort and defects.

Fig. 3 shows the market and sales sector accounting for market share dynamics and software license sales. The perceived quality is a reputation factor that can reduce the number of sales if products have many defects (see the next section).

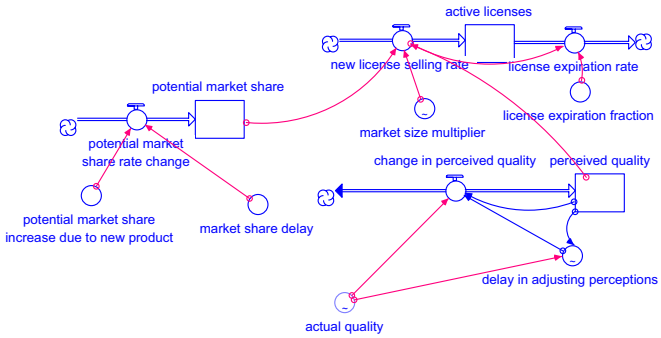


Fig. 3. Market and sales sector

The market and sales model presented herein is a simplification of a more extensive being used in industry that accounts for additional marketing initiatives and software license maintenance sales.

The finance sector is shown in Fig. 4. It includes cash flows for investment and revenue. Investments include the labor costs for software development, maintenance and associated activities.

Revenue is derived from the number of license sales. Sales are a function of the overall market size and market share percentage for the software product. The market share is computed using a potential market share adjusted by

perceived quality. The additional market share derivable from a new product is attained at an average delay time. More details of the model are provided in [4].

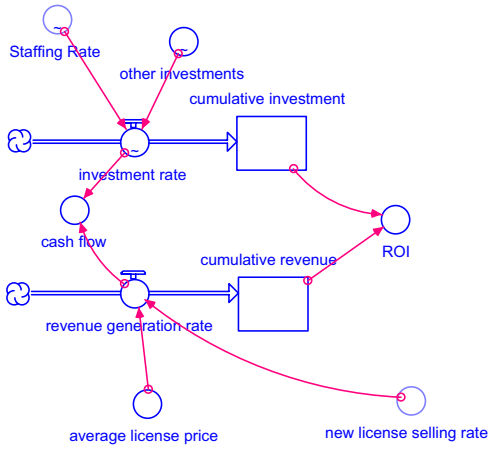


Fig. 4. Finance sector

A. *Quality Modeling and Value Functions*

For simplification, software reliability as defined in the COCOMO II model [5] is used as a proxy for all quality practices. It models the tradeoff between reliability and development cost. There are four different settings of reliability from low to very high that correspond to four development options. The tradeoff is increased cost and longer development time for increased quality. This simplification can be replaced with a more comprehensive quality model (see Conclusions and Future Work).

The resulting quality will modulate the actual sales relative to the highest potential. A lower quality product will be done quicker; it will be available on the market sooner but sales will suffer from poor quality. The mapping between reliability and the relative impact to sales from continuing Delphi surveys is now captured as a production function and used in the model with the latest refined numbers.

Collectively there are two value-based production functions in the model to describe value relationships (they are illustrated in the first applied example). A market share production function addresses the organizational business value of product features. The business value is quantified in terms of added potential market share attainable by the features. The relationship assumes that all features are implemented to the highest quality. Since the required reliability will impact how well the features actually work, the relationship between reliability costs and actual sales and is needed to vary the sales due to quality.

The value function for actual sale attainment is relevant to two classes of stakeholders. It describes the value of different reliability levels in terms of sales attainment, and is essentially a proxy for user value as well. It relates the percent of potential sales attained in the market against reliability costs. Illustrations of the production functions are shown in the next section.

The market and sales sector also has a provision to modulate sales based on the perceived quality reputation. A bad quality reputation takes hold almost immediately with a buggy product (bad news travels fast), and takes a long time to recover from in the market perception even after defects are fixed. This phenomenon is represented with asymmetrical information smoothing as shown in Fig. 5 with a variable delay in adjusting perceptions.

The graph in Fig. 5 shows a poor quality release at time=3 years with a followup release to the previous quality level. While the perceived quality quickly plummets after the bad release, it rises much more slowly even when the actual quality has improved.

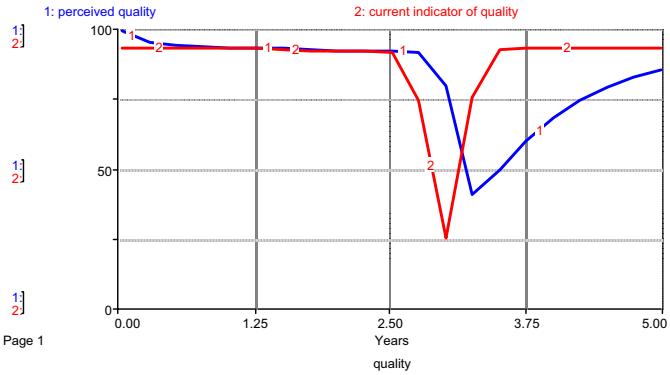


Fig. 5. Perceived quality trends with high and low quality product deliveries

III. Applied Examples

Several representative business decision scenarios are demonstrated in this section. The first one demonstrates the ability to dynamically assess combined strategies for scope and reliability. The second example looks at strategies of multiple releases of varying quality. Finally the model is used to determine a process sweet spot for reliability.

A. *Dynamically Changing Scope and Reliability*

The model can be used to assess the effects of individual and combined strategies for overall scope and reliability. This example will show how it can be used to change product specifications midstream as a re-plan. Static cost models typically do not lend themselves to re-plans after the project starts, as all factors remain constant through time. This dynamic capability can be used in at least two ways by a decision-maker:

- assessing the impact of changed product specifications during the course of a project
- before the project starts, determining if and how late during the project specifications can be changed based on new considerations that might come up.

Three cases are simulated: 1) an unperturbed reference case, 2) a midstream descoping of the reference case and 3) a simultaneous descoping and lowered required reliability. Such descoping is a frequent strategy to meet time constraints by shedding features.

The market share production function in Fig. 6 relates the potential business value against the cost of development for different feature sets. The actual sales production function against reliability costs is shown in Fig. 7, and it is applied against the potential market capture. The four discrete points correspond to required reliability levels of low, nominal, high and very high. Settings for the three cases are shown in both production functions.

Fig. 8 shows a sample control panel interface to the model. The primary inputs for product specifications are the size in function points (also called scope) and required reliability. The number of function points is the size to implement given features. The size and associated cost varies as the number of features to incorporate.

The reliability settings on the control panel slider are the relative effort multipliers to achieve reliability levels from low to very high. These are input by user via the slider for "Reliability Setting". The attainable market share derived from the production function in Fig. 6 is input by the user on the slider "Potential Market Share Increase".

Fig. 8 also shows the simulation results for the initial reference case. The default case of 700 function points is delivered with nominal reliability at 2.1 years with a potential 20% market share increase. This project is unperturbed during its course and the 5 year ROI of the project is 1.3.

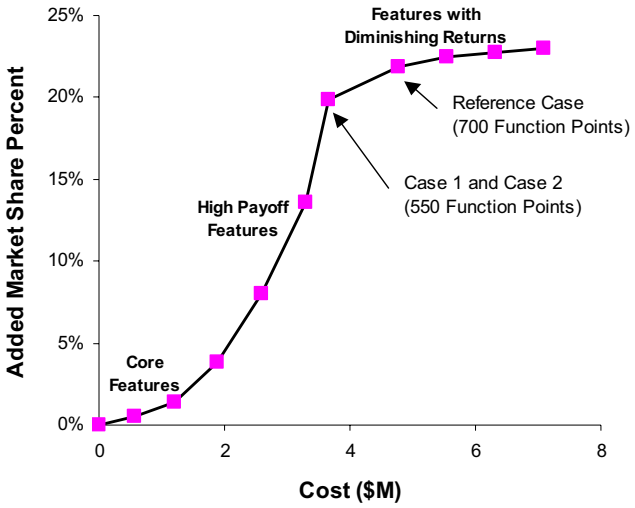


Fig. 6. Market share production function and feature sets

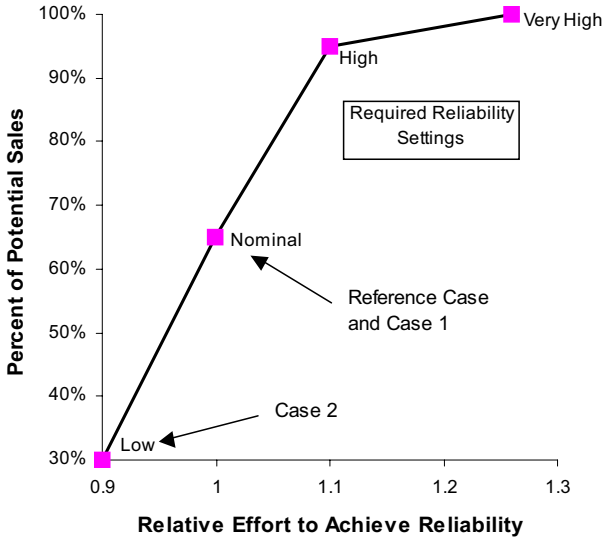


Fig. 7. Sales production function and reliability

Case 1 in Fig. 9 illustrates the initial case perturbed at .5 years to descope low-ROI features (see Fig. 6 and Fig. 7 for the points on the production function). The scope goes down to 550 function points and the staffing profile adjusts dynamically for it. The schedule is reduced by a few months. In this case the potential market share increase is lowered by only two percentage points to 18%. With lower development costs and earlier delivery the ROI increases substantially to 2.2.

A combined strategy is modeled in Fig. 10 for Case 2. The scope is decreased the same as before in Case 1 (Fig. 9) plus the reliability setting is lowered from nominal to low. Though overall development costs decrease due to lowered reliability, the market responds poorly. This case provides the worst return of the three options and market share is lost instead of gained.

In Case 2 there is an early hump in sales due to the initial hype of the brand new product, but the market soon discovers the poor quality and then sales suffer dramatically. These early buyers and others assume the previous quality of the product line and are anxious to use the new, "improved" product. Some may have pre-ordered and some are early adopters that always buy when new products come out. They are the ones that find out about the lowered quality and the word starts spreading fast.

A summary of the three cases is shown in Table I. Case 1 is the best business plan to shed undesirable features with diminishing returns. Case 2 severely hurts the enterprise because quality is too poor.

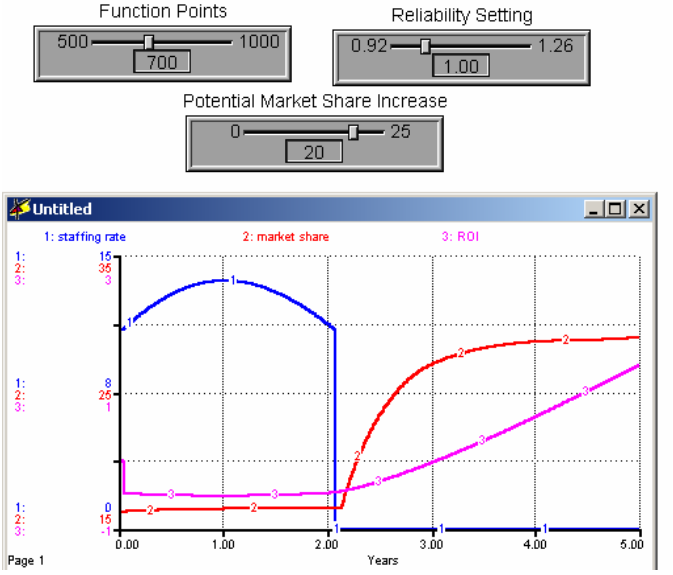


Fig. 8. Sample control panel and reference case (unperturbed)

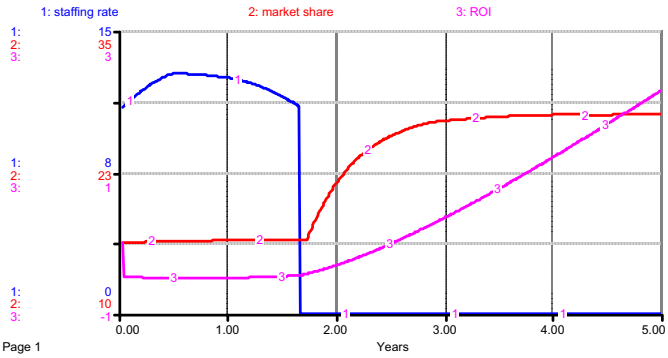


Fig. 9. Case 1 – descoping of low ROI features at time = .5 years

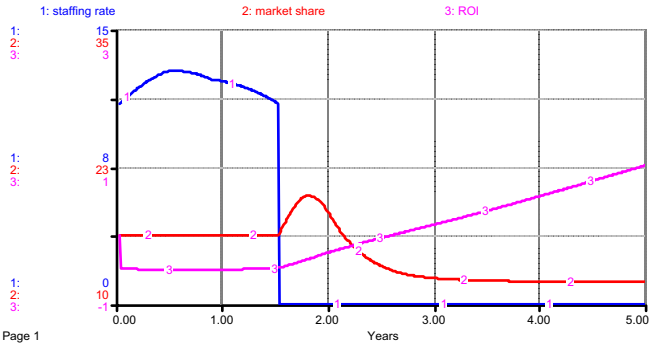


Fig. 10. Case 2 – descoping of low ROI features and reliability lowering at time = .5 years

TABLE I
CASE SUMMARIES

Case	Delivered Size (Function Points)	Delivered Reliability Setting	Cost ($M)	Delivery Time (Years)	Final Market Share	ROI
Reference Case: Unperturbed	700	1.0	4.78	2.1	28%	1.3
Case 1: Descope	550	1.0	3.70	1.7	28%	2.2
Case 2: Descope and Lower Reliability	550	.92	3.30	1.5	12%	1.0

B. Multiple Releases

This example shows a more realistic scenario for maintenance and operational support. Investments are allocated to ongoing maintenance and the effects of additional releases of varying quality are shown.

The reference case contains two product rollouts at years 1 and 3, each with the potential to capture an additional 10% of the market share. These potentials are attained because both deliveries are of high quality as seen in Figs. 11-12.

A contrasting case in Figs. 13-14 illustrates the impact if the second delivery has poor quality yet is fixed quickly (Fig. 5 shows the quality trends for this case). This results in a change of revenue from $11.5 M to $9.6M, ROI from 1.3 to 0.9.

This example is another illustration of the sensitivity of the market to varying quality. Only one poor release in a series of releases may have serious long term consequences.

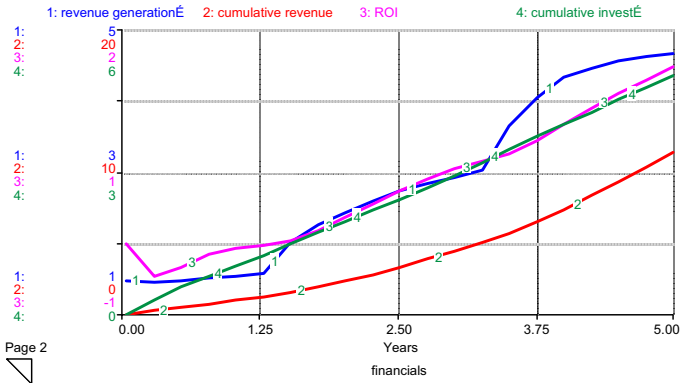


Fig. 11. Reference case financials for two high quality product deliveries

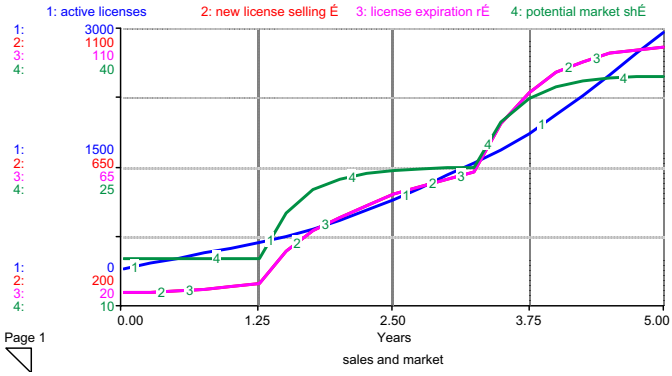


Fig. 12. Reference case sales and market for two high quality product deliveries

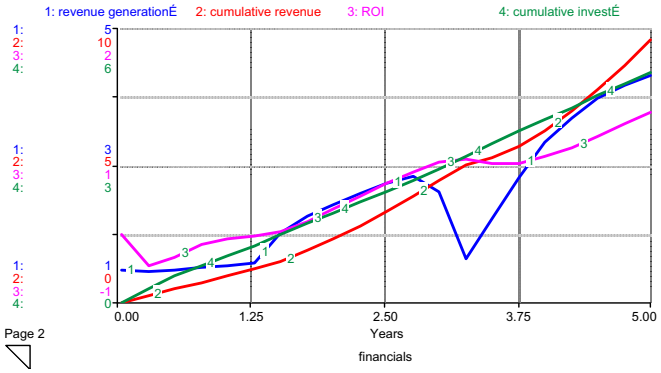


Fig. 13. Financials for high and low quality product deliveries

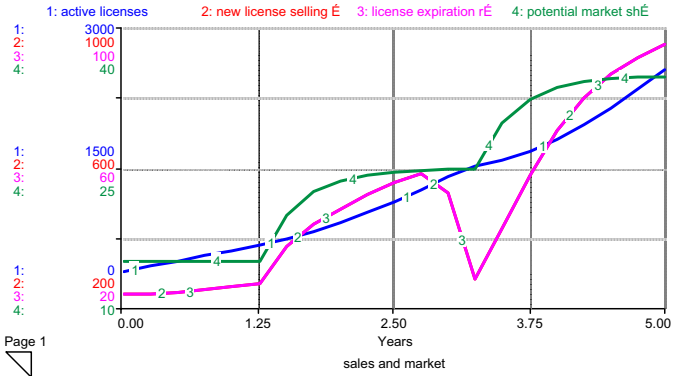


Fig. 14. Sales and market for high and low quality product deliveries

C. *Finding the Sweet Spot*

This example derived from [4] shows how the value-based product model can support software business decision-making by using risk consequence to find the quality sweet spot with respect to ROI. The following analysis steps are performed to find the process sweet spot:

- vary reliability across runs
- assess risk consequences of opposing trends: market delays and bad quality losses
- sum market losses and development costs
- calculate resulting net revenue to find process optimum.

The risk consequences are calculated for the different options. Only point estimates are used for the sake of this example. A more comprehensive risk analysis would consider probability distributions to obtain a range of results. Probability is considered constant for each case and is not explicitly used in the calculations. Only the costs (or losses) are determined.

A set of runs is performed that simulate the development and market release of a new 80 KSLOC product. The product can potentially increase market share by 30%, but the actual gains depend on the level of quality. Only the highest quality will attain the full 30%. Other parameterizations are an initial total market size = $64M annual revenue, the vendor has 15% initial market share, and the overall market doubles in 5 years.

A reference case is needed to determine the losses due to inferior quality. The expected revenues for a sub-quality delivery must be subtracted from the maximum potential revenues (i.e. revenue for a maximum quality product delivered at a given time). The latter is defined as delivering a maximum quality product at a given time that achieves the full potential market capture. The equation for calculating the loss due to bad quality is

Bad Quality Loss = Maximum Potential Revenue with Same Timing – Revenue.

The loss due to market delay is computed keeping the quality constant. To neutralize the effect of varying quality, only the time of delay is varied. The loss for a given option is the difference between the revenue for the highest quality product at the first market opportunity and the revenue corresponding to the completion time for the given option (assuming the same highest quality). It is calculated with

Market Delay Cost = Maximum Potential Revenue – Revenue.

Fig. 15 shows the experimental results for an 80 KSLOC product, fully compressed development schedules and a 3-year revenue timeframe for different reliability options. The resultant sweet spot corresponds to reliability=high. The total cost consisting of delay losses, reliability losses and development cost is minimum at that setting for a 3-year time horizon. Details of the intermediate calculations for the loss components are provided in [4].

The sweet spot depends on the applicable time horizon, among other things. The horizon may vary due for several reasons such as another planned major upgrade or new release, other upcoming changes in the business model, or because investors mandate a specific timeframe to make their return.

The experiment was re-run for typical time horizons of 2, 3 and 5 years using a profit view (the cost view is transformed into a profit maximization view by accounting for revenues). The results are shown in Fig. 16.

The figure illustrates that the sweet spot moves from reliability equals low to high to very high. It is evident that the optimal reliability depends on the time window. A short-lived product (a prototype is an extreme example) does not need to be developed to as stringent reliability as one that will

live in the field longer.

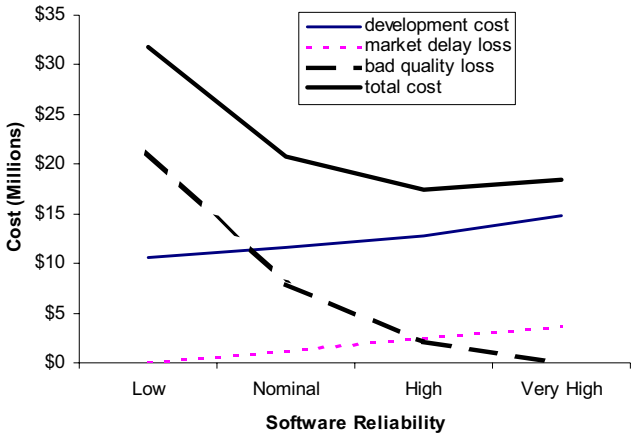


Fig. 15. Calculating reliability sweet spot (3-year timeframe)

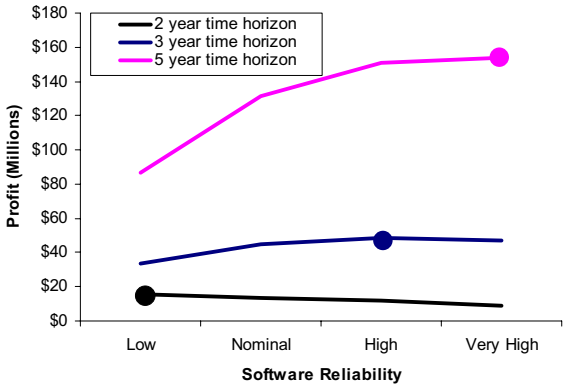


Fig. 16. Reliability sweet spot as a function of time horizon

IV. Conclusions and Future Work

It is crucial to integrate value-based methods into the software engineering discipline. To achieve real earned value, business value attainment must be a key consideration when designing software products and processes. This work shows several ways how software business decision-making can improve with value information gained from simulation models that integrate business and technical perspectives.

The model demonstrates a stakeholder value chain whereby the value of software to end users ultimately translates into value for the software development organization. It also illustrates that commercial process sweet spots with respect to reliability are a balance between market delay losses and quality losses. Quality does impact the bottom line.

The model can be elaborated to account for feedback loops to generate revised product specifications (closed-loop control). This feedback includes:

- external feedback from user to incorporate new features
- internal feedback on product initiatives from an organizational planning and control entity to the software process.

A more comprehensive model would consider long term product evolution and periodic upgrades. Another related aspect to include is general maintenance by adding explicit activities for operational support.

The product defect model can be enhanced with a dynamic version of COQUALMO [6] to enable more constructive insight into quality practices. This would replace the current construct based on the single factor for required software reliability.

Other considerations for the model are in the market and sales sector. The impact of different pricing schemes and varying market assumptions on initial sales and maintenance can all be explored. Some of these provisions are already accounted for in a proprietary version of the model.

The model application examples were run with idealized inputs for sake of demonstration, but more sophisticated dynamic scenarios can be easily handled to model real situations. For example discrete descopings were shown, but in many instances scope will exhibit continuous or fluctuating growth over time.

More empirical data on the relationships in the model will also help identify areas of improvement. Assessment of overall dynamics includes more collection and analysis of field data on business value and quality measures from actual software product rollouts.

References

[1] B. Boehm, L. Huang: "Value-based software engineering: A case study", *IEEE Software*, Vol. 20, No. 2, 2003

[2] B. Boehm, L. Huang, A. Jain, R. Madachy, "Reasoning about the ROI of software dependability: the iDAVE Model", *IEEE Software*, Vol. 21, No. 3, 2004

[3] R. Madachy, "A software product business case model", Proceedings of the 5th International Workshop on Software Process Simulation and Modeling, 2004

[4] R. Madachy, *Software Process Dynamics*, IEEE Computer Society Press, Washington D.C., 2005 (to be published)

[5] B. Boehm, C. Abts, W. Brown, S. Chulani, B. Clark, E. Horowitz, R. Madachy, D. Reifer, B. Steece, *Software Cost Estimation with COCOMO II*, Prentice-Hall, 2000

[6] S. Chulani, B. Boehm, "Modeling software defect introduction and removal: COQUALMO (COnstructive QUALity MOdel)", USC-CSE Technical Report 99-510, 1999

A Software Product Line Process Simulator

Yu Chen, Gerald C. Gannod, and James S. Collofello

Abstract—A software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission, and are developed from a common set of core assets in a prescribed way. A software product line approach promises shorter time-to-market and decreased life cycle cost. However, those benefits are not guaranteed under every situation and are affected by many factors, such as number of available employees, market demands, reuse rate, process maturity, and product line adoption and evolution approaches. Before initiating a software product line, an organization needs to evaluate available process options in order to see which one best fits its goals. The aim of this research is to develop a software product line process simulator that can predict the cost for a selected software product line process and provide useful information for cost-benefit analysis.

I. Introduction

A software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission. A defining characteristic of product lines is that products are developed from a common set of core assets in a prescribed way [1]. A software product line approach promises shorter time-to-market, higher product quality, and decreased life cycle cost [1]. However, those benefits are not guaranteed under every situation and are affected by many factors including the number of available employees, market demands, reuse rate, process maturity, and the selected product line adoption and evolution approaches. Before initiating a software product line, an organization needs to evaluate available process options in order to see which one best fits its goals. The aim of this research is to develop a software product line process simulator that can predict the cost for a selected software product line process and provide useful information for cost-benefit analysis. Several techniques have used simulation to study software processes, including [2]. However, to our best knowledge, this is the first simulator in literature to address software product line process issues.

In this paper, an architecture-based software product line process simulator is presented. The simulator uses Microsoft Project [3] to define an organization's product line development process. DEVSJAVA [4] is used as the modeling and simulation formalism and COPLIMO [5] is used as the underlying cost model. The simulator is meant to be used after the high-level product line and product architectures have been defined. The inputs to the simulator include a product line life cycle project plan at the component granularity level, available resources, product demands, and cost model parameter values. These inputs are at a level that allows for organizational level product line planning. The outputs from the simulator provide estimates of the first release time, initial development effort, life cycle effort for each product, life cycle effort for the whole product line, and resource usage rates. By varying the inputs and comparing the outputs, a manager can make decisions about whether a product line approach should be used, and given that one is used, what strategies should be adopted, and what the potential resource allocations should be.

In our previous work, we developed an early stage software product line simulator [6] meant to be used when the product line architecture and system features are still vague. As such, it makes some simplification assumptions about the uniformity of the product size, change rate, reuse rate, etc. Also, it only supports a limited number of product line adoption and evolution strategies. The architecture-based simulator presented in this paper is meant to be used when the product line architecture, feature model, and product map are well-defined. It removes many simplifying assumptions made by the previous simulator, supports more general product line processes, and provides more information in the simulation results.

The remainder of the paper is organized as follows. Section 2 presents background information and related work. Section 3 describes the approach and the simulator. Results are discussed in Section 4. Section 5 draws conclusions and suggests future investigations.

II. Background and Related Work

This section describes background and related work on software product lines and product line cost models.

A. Software Product Lines

Software product line development involves three essential activities: core asset development, product development, and management. Core asset development (domain engineering) involves the creation of common assets and the evolution of the assets in response to product feedback, new market needs, etc. Product development (application engineering) creates individual products by reusing the common assets, gives feedback to core asset development, and evolves the products. Management includes technical and organizational management, where technical management is responsible for requirement control

This material is based upon work supported by the National Science Foundation under career grant No. CCR-0133956.

Yu Chen, Dept. of Computer Science and Engineering, Arizona State University - Tempe Campus, Tempe AZ 85287, USA, yu_chen@asu.edu.

Gerald C. Gannod, Division of Computing Studies, Arizona State University - East Campus, Mesa AZ 85212, USA, gannod@asu.edu.

James S. Collofello, Dept. of Computer Science and Engineering, Arizona State University - Tempe Campus, Tempe AZ 85287, USA, collofello@asu.edu.

and the coordination between core asset and product development.

A software product line can be initiated under several situations: *independent*, *project-integration*, *reengineering-driven*, and *leveraged* [7]. Under the independent situation, a product line is created without any pre-existing products. Under the project-integration situation, a product line is created to support both existing and future products. Under a reengineering-driven scenario, a product line is created by reengineering existing legacy systems. And the leveraged situation is where a new product line is created based on some existing product lines.

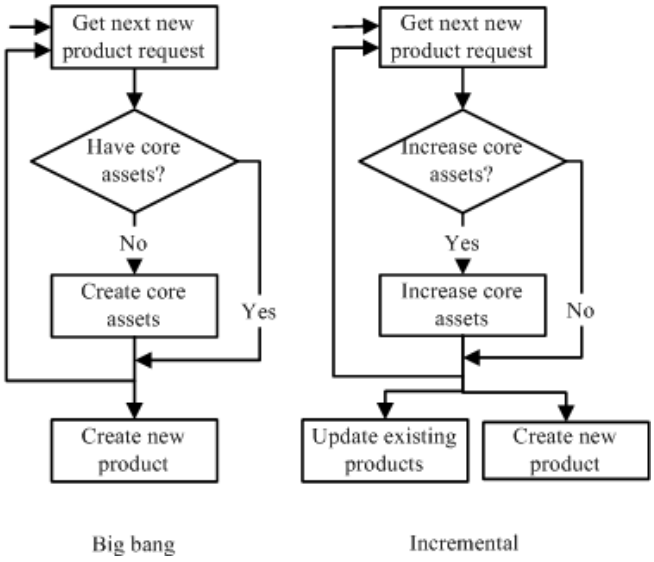


Fig. 1. Proactive approach process flow

There are two main software product line development approaches: *proactive* and *reactive*. With the proactive approach, core assets are developed first to support future products. With the reactive approach, core assets are incrementally created when new requirements arrive. Depending on the degree of planning-ahead, the proactive approach can be classified into *big bang* and *incremental* approach [7]. With the big bang approach, core assets are developed for a whole range of products prior to the creation of any individual product. With the incremental approach, core assets are incrementally developed to support the next few upcoming products. Fig. 1 shows the process flow of the proactive approaches. Some common reactive approaches are: *infrastructure-based*, *branch-and-unite*, and *bulk-integration* [7]. The infrastructure-based approach does not allow deviation between the core assets and the individual products, and requires that new common features be first implemented into the core assets and then built into products. Both the branch-and-unite and the bulk-integration approaches allow temporal deviation between the core assets and the individual products. The branch-and-unite strategy requires that the new common features be reintegrated into the core assets immediately after the release of the new product, while the bulk-integration strategy allows the new common features to be reintegrated after the release of a group of products. Fig. 2 shows the process flow for the infrastructure-based and branch-and-unite approach. These approaches are not mutually exclusive. For instance, a product line can be adopted via a proactive approach and then evolved through a reactive approach.

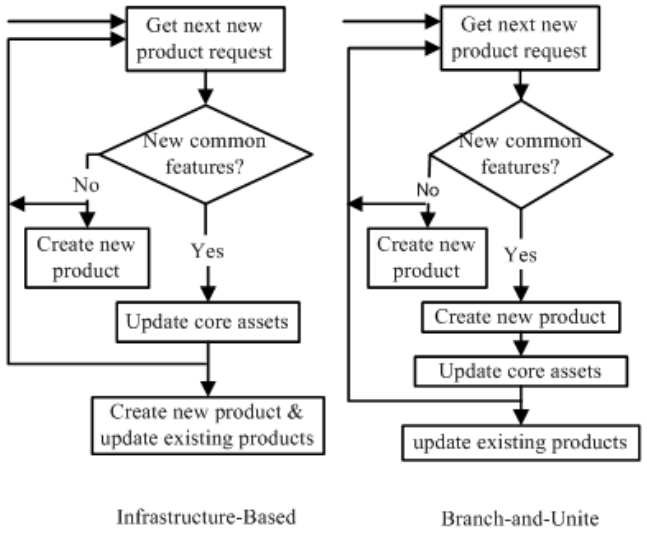


Fig. 2. Reactive approach process flow

B. *Product line cost models*

Several software product line cost estimation approaches [5], [10], [11] have been proposed. Bockle *et al.* discussed software product line adoption scenarios and presented a product line cost model [10]. Among the seven adoption scenarios, only two of them, developing a single product line without pre-existing products, are considered by this paper. Their cost model takes organization costs into account, which is not considered in this work. Boehm *et al.* proposed COPLIMO [5], a COCOMO II [12] based model, for software product line cost estimation. COPLIMO has a basic life cycle model, which consists of a product line development cost model and an annualized post-development extension.

This simulator uses COPLIMO [5] as the underlying cost model. However, in the implementation, the cost model is designed as a plug-in model, thus allows other cost models to be used as well. The early stage simulator [6] uses the COPLIMO basic life cycle model. To allow more detailed modeling, the basic life cycle model is extended by using COCOMO II [12] and is used by this architecture-based simulator.

III. Approach

This section presents the overview of our approach, the software product line process model, and the simulation tool.

A. *Overview*

Before initiating a software product line, an organization needs to evaluate available process options in order to see which one best fits its goals. Fig. 3 shows the process evaluation steps and where the simulator fits into the process. Software product line process is first defined by using architecture definition, then the process definition is simulated, and the simulation results are analyzed. The current tools used for these steps are Microsoft Project [3],

the simulator discussed here, and Microsoft Excel, respectively. The cost, time, and resource usage estimates provided by the simulator can also be used to refine the process definition. For instance, the outputs can be reinterpreted into process definition to provide a detailed project schedule plan.

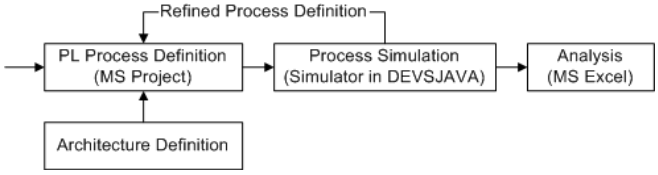


Fig. 3. Process evaluation

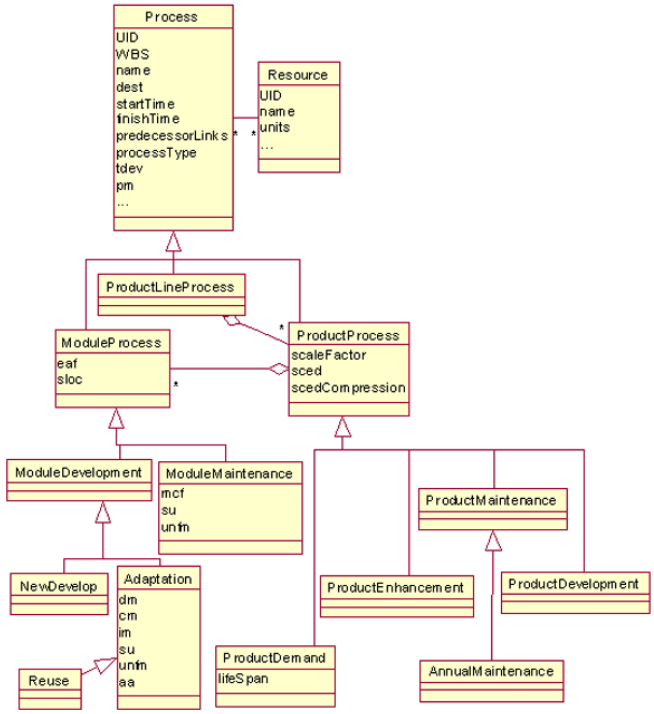


Fig. 4. Process meta-model

B. *Software product line process model*

The organizational level software product line development process was abstracted into a model as shown in Fig. 4. This model categorizes processes into three levels *product line*, *product*, and *module*. A product line process consists of several product processes. Each product process is one of the following types: *product demand*, *product development*, *product maintenance*, *product enhancement*, and *annual maintenance*. Product demand represents external new product requirements. Product development refers to initial product creation. Product maintenance represents product modification activity, and annual maintenance refers to annually planned product maintenance. Product enhancement models the maintenance activity that is to improve the product functionality and involves major product change. It is not considered as product maintenance because it often involves different cost calculation method. A product process has several module processes. Each module process is either a *module development* or *maintenance* process. A module development process is one of the following types: *new development*, *adoption*, and *reuse*. New development means development without reuse, adoption refers to white-box reuse, and reuse means black-box reuse. Module maintenance represents module modification activities.

Some attributes associated with the models are shown in Fig. 5. The attributes include standard Microsoft Project parameters (e.g. *UID*, *WBS*, and *predecessorLinks*), parameters needed by the cost model [5] (e.g. *dm*, *cm*, and *im*), and simulator specific parameters (e.g. *processType*). The temporal relationships among processes are described by ``predecessorLinks", which is a list of ``predecessorLink". If Process *A* has a ``predecessorLink" that points to Process *B*, then Process *A* can not start until Process *B* is finished. Each process can have resources associated with it. For this study, only the human resources to conduct the processes are considered.

Attribute	Description
UID	Unique identification
WBS	Work breakdown structure code
startTime	Process start time
finishTime	Process finish time
predecessorLinks	A list of links to the predecessors
processType	The type of the process as described above
dest	The target of the process, such as the name of a product line, product, and module
tdev	Process duration in months
pm	Effort in person-months
units	The number of resources
scaleFactor	The sum of scale drivers
sced	Project schedule
scedCompression	Project schedule compression
lifeSpan	Product life span in years
eaf	The product of effort multipliers
sloc	Source line of code
mcf	Maintenance change factor
dm	Percentage of design modification
cm	Percentage of code modification
im	Percent of Integration Required for Modified Software
su	Percentage of reuse effort due to software understanding
unfm	programmer unfamiliarity with software
aa	percentage of reuse effort due to assessment and assimilation

Fig. 5. Attribute description

To define a software product line process of this kind, inputs from market analysis, high-level feature model and product map, and product line development and evolution strategies are needed. Market analysis results tell what kind of products will be needed in the future and when they will be needed. The feature model and product map allows decomposing products into high-level modules and recognizing reusable modules. The product line development and evolution strategies provide guidance on how to create and evolve core assets as well as products and in which order. This software product line process is mainly for organizational level management control, so only one level module process is provided. If more detailed management is needed, the process can serve as process architecture and allow further module level process refinement.

C. *Simulation tool*

The simulator is implemented in Java using the DEVSJAVA [4] formalism. It uses Microsoft Project [3] as the process definition tool and COPLIMO [5] as the cost model. The input is a process definition (in XML format) that conforms to the previously discussed process meta-model. In the process definition, only the resources for the product line level processes need to be specified. The simulator will use the cost model to assign resources to the lower level processes.

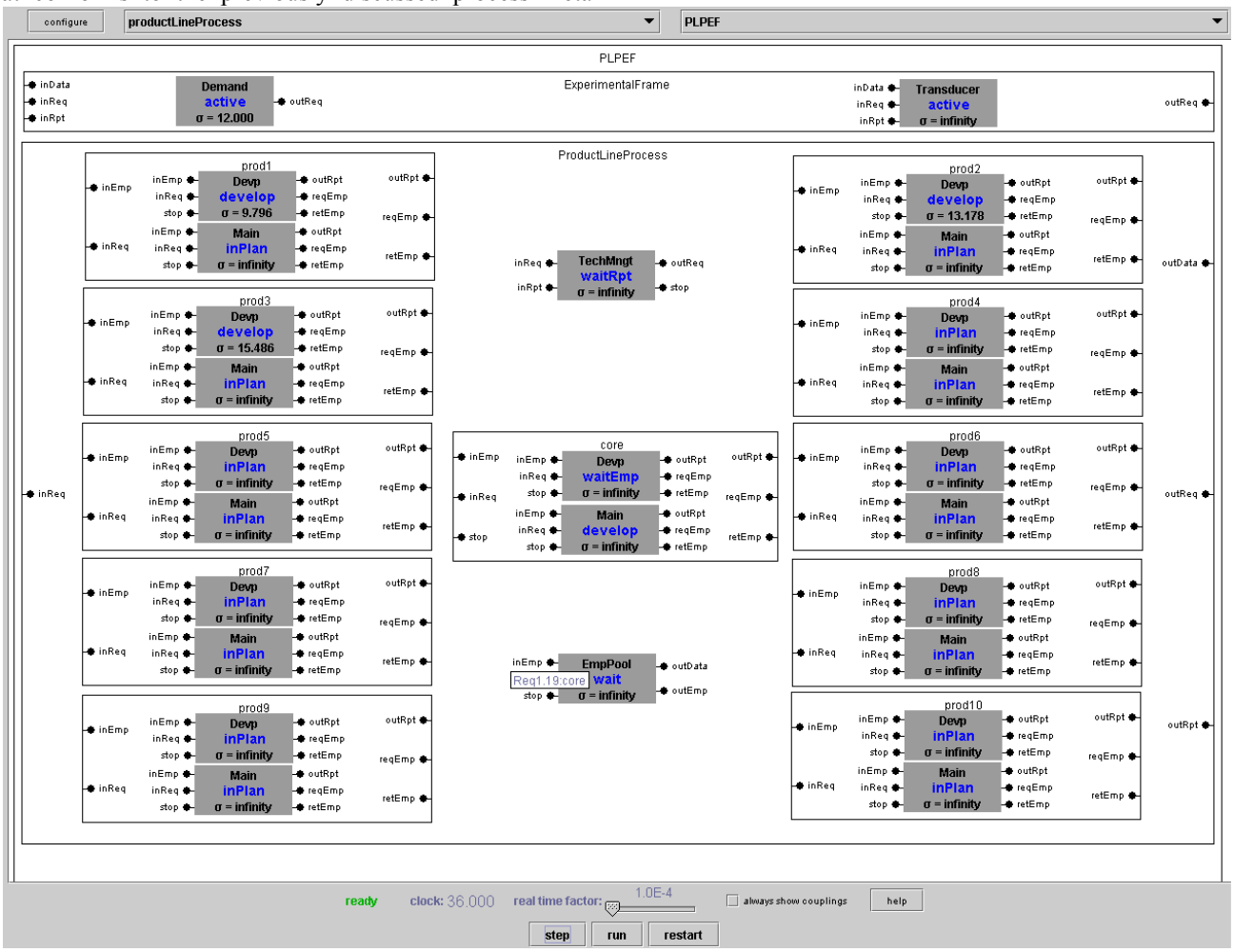


Fig. 6. Simulation tool in execution

Product Section								
ProdName	ISLOC	FRT	TTM	IDE	IDT	ADE	ADT	AWT
core	160000	35.13	35.13	1217.15	35.13	5611.47	312.3	34.56
prod1	100000	45.8	45.8	28.7	10.67	53.28	136.67	16.41
prod2	130000	49.18	37.18	68.23	14.05	135.48	140.05	23.83
prod3	160000	51.49	27.49	110.08	16.36	224.79	148.36	17.83
prod4	170000	98.74	62.74	58.96	13.41	95.86	139.41	5.05
prod5	230000	103.06	55.06	142.54	17.73	243.47	143.73	6.37
prod6	290000	118.1	58.1	233.03	20.77	404.03	152.77	75.18
prod7	130000	111.38	39.38	68.23	14.05	111.35	134.05	0
prod8	160000	121.11	37.11	110.08	16.36	184.79	139.36	6.72
prod9	230000	128.54	32.54	143.45	17.8	235.75	137.8	0
prod10	290000	135.15	27.15	233.03	20.77	387.66	140.77	0
Product Line Section								
Name	ADE	ADT	LS	AADE	ATTM	AWT		
prodLine	7687.93	1725.27	255.15	361.57	42.25	185.96		
Resource Section								
Name	PWT	AUR	MinUR	MaxUR				
EmpPool	0.13	0.55	0.28	1				

Fig. 7. Simulation results

The simulator user interface is depicted in Fig. 6. The upper part of the interface identifies the current running model and its package ("PLPEF" and "productLineProcess", respectively). The large middle area shows the models and their hierarchical relationships. The bottom of the window contains execution control components. The "step" button allows running the simulator step by step, the "run" button allows executing the simulator to the end, and the "restart" button allows starting a new simulation run without quitting the system. The "clock" label displays the current simulation time in the unit of months, and selecting the "always show couplings" checkbox will allow couplings between models to be displayed. The simulation speed can be manipulated at run time to allow execution in near real-time or logical time (slower/faster than real-time).

At the end of each simulation run, a result table is generated similar to Fig. 7. The table has three sections. The product section provides data related to individual products including initial source line of code (ISLOC), first release time (FRT), time-to-market (TTM), initial development effort (IDE), initial development time (IDT), accumulated development and maintenance effort (ADE), accumulated development and maintenance time (ADT), and accumulated process waiting time (AWT). The product line section summarizes the product line related statistics, which include accumulated development and maintenance effort (ADE), accumulated development and maintenance time (ADT), product line life span (LS), average annual development effort (AADE), average time-to-market (ATTM), and accumulated process waiting time (AWT). In the table, the unit of effort is person-months and the unit of time is months. The resource section provides resource usage data including percentage of the time the resource pool is in the wait stage for the lack of resources (PWT), average resource usage rate (AUR), minimum resource usage rate (MinUR) , and maximum resource usage rate (MaxUR). Comparing with the early stage simulator result, this result adds two sections (Product Line and Resource) and two fields (ISLOC and AWT) for the Product section.

The following assumptions are made in the simulation model:

1. All the employees have the same capability and can work on any project.
2. Every product (including core asset) has two concurrent development processes: one for planned maintenance and one for development and unplanned maintenance. Each of the process handles the requirements in FIFO order.
3. For each product, its fractions of new product-specific, adapted, and reused code stay relatively constant across its life cycle.

Assumption 1 is a simplification of the reality, where each employee has different skills and capability. The purpose of this simulator is to give high-level cost and time estimates, so the average is used to model employee capability. If detailed modeling is needed, more *Employee Pool* instances can be used to model employees with different level of capabilities. Assumption 2 is based on experience with software development in real development organizations. In the case that there are more concurrent processes for a product, more *Development* or *Maintenance* instances can be used. Assumption 3 is made by the cost model, COPLIMO [5], and can be changed if other cost models are used.

IV. Results

In this section a case study is presented to illustrate the use and the analytical capability of the simulator. The data shown is the result of executing the simulation system.

A. Overview

The target of the case study is a simulator product line. Specifically, the case study models a product line of software process simulators. Its feature model [13] is depicted in Fig. 8. The feature model indicates that a simulator must have an IO, Cost Model, and Simulation Model. The IO can include a basic IO, and optionally Web or GUI IO. The Cost Model can use either Model1 or Model2. The Simulation Model can include Early Stage model, or Later Stage model, or both. Fig. 9 provides the name, description, and size information for each high-level component within the product line. The product line product map displayed in Fig. 9 shows the relationships between the components and individual products. The feature model, high-level component map, and product map are the results of market analysis and high-level architecture design. Based on this information and market demand analysis, development strategies and decisions (such as which components should be included in the core assets, how and core assets should be developed, evolved, and reused, and when the products should be developed and maintained) can be represented by using the product line process model described earlier.

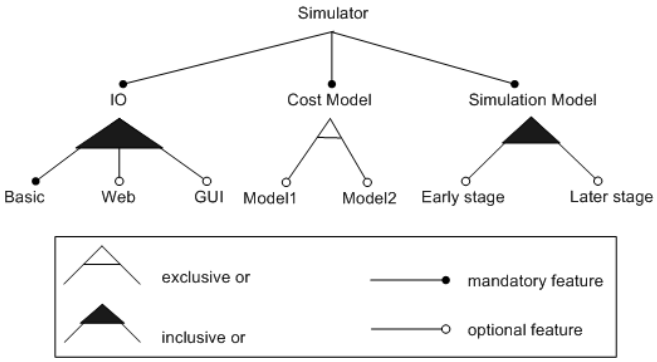


Fig. 8. Feature model

Eight different strategies for building the product line were defined into eight processes and then simulated. The first seven strategies all use the product line approach, while the last strategy uses traditional independent product development approach. The product line strategies differ in the number of employees, adoption approaches, and evolution methods. They all use proactive approach (big bang or incremental) for product line adoption and reactive approach (infrastructure-based or branch-and-unite) for product line evolution. For the big bang approach, the core assets are fully developed before Product 1. For the incremental approach, the core assets are first developed before Product 1 and then increased before Product 4.

Name	Description	Size
C1	Basic IO - early stage	20k
C2	Web IO - early stage	30k
C3	GUI IO - early stage	30k
C4	Basic IO - later stage	20k
C5	Web IO - later stage	30k
C6	GUI IO - later stage	30k
C7	Simulation model - early stage	50k
C8	Simulation model - later stage	50k
C9	Cost model1	30k
C10	Cost model2	30k

Fig. 9. High-level components

Name	Prod1	Prod2	Prod3	Prod4	Prod5	Prod6	Prod7	Prod8	Prod9	Prod10
C1	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
C2		✓	✓		✓	✓	✓	✓	✓	✓
C3			✓			✓		✓		✓
C4				✓	✓	✓			✓	✓
C5					✓	✓			✓	✓
C6						✓				✓
C7	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
C8				✓	✓	✓			✓	✓
C9	✓	✓	✓	✓	✓	✓				
C10							✓	✓	✓	✓

Fig. 10. Product map

	Strategy							
Parameter	1	2	3	4	5	6	7	8
Number of employees	50	55	60	90	90	90	90	90
Big bang				✓		✓		
Incremental	✓	✓	✓		✓		✓	
Infrastructure based	✓	✓	✓	✓	✓			
Branch and unite						✓	✓	
Independent development								✓

Fig. 11. Strategies

B. *Effect of resources*

Strategies 1, 2, 3 and 5 differ in the number of employees (50, 55, 60, and 90, respectively) in the context of incremental adoption and infrastructure-based evolution. The results of their simulations differ in time-to-market, as shown in Fig. 12, which was generated using data similar to that shown in Fig. 7. For the first three products, there is no difference in time-to-market, because the available resources are sufficient for all the three cases. Starting from Product 4, we see more resource-constrained strategies have longer time-to-market than less resource-constrained ones. When the request for Product 4 comes, more resources are needed for increasing the core assets while some resources are still held by new product development activities. For the processes that do not have enough resources, their core asset incremental activities are put on hold until more resources are made available, which in turn delays development of later products. Another reason is that as more products are developed, more resources are used for maintenance and fewer resources are available for new product development. In the case that there are not enough resources, processes with fewer resources need to wait longer to start the development or maintenance activities. Fig. 13 shows that Percentage of Waiting Time for these strategies. We can see that as the number of employees reduces the percentage of waiting time increases, which explains the time-to-market differences between these processes. Strategy 5 provides sufficient resources for the product line development, so its percentage of waiting time is 0. By comparing the results, a manager can make decision on how many resources should be allocated to the product line development.

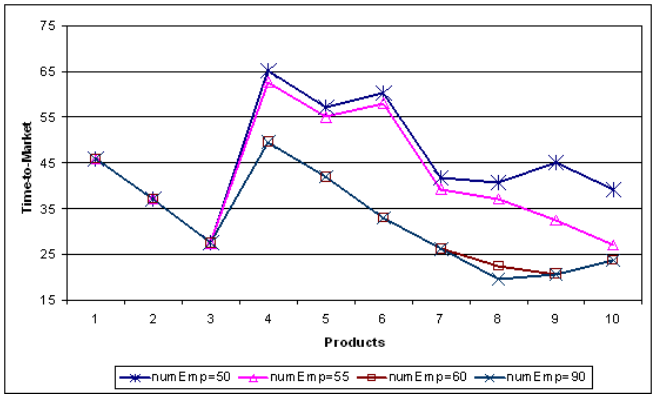


Fig. 12. Effect of resources on time-to-market

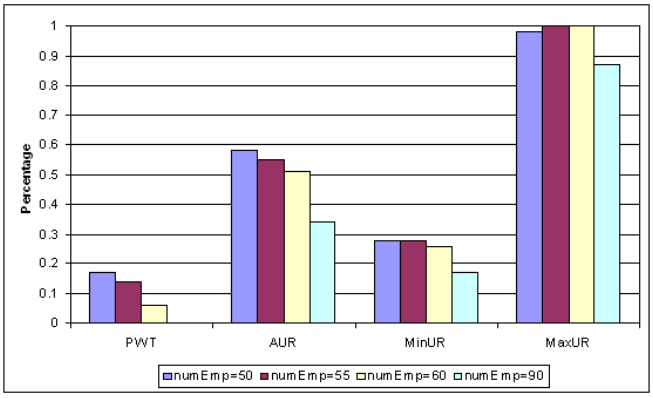


Fig. 13. Effect of resources on resource usage

Figure 13 shows that there is a downward trend of time-to-market after Product 4, because the resource request peak happens when the core assets need to be increased. After that resource requests tend to decline. In our previous study [6] on the effect of resources, the time-to-market had an increasing trend in the context of big bang adoption and infrastructure-based evolution. That is because after the initial core development, the resource request peak gradually comes when more products are under maintenance while new products need to be developed. In general, lack of resources will result in longer time-to-market, but the trend of time-to-market over the time depends on the characteristics of the individual processes, such as adoption and evolutions strategies, reuse rates, and the frequency of market demands.

C. *Effect of evolution approaches*

Strategies 4 and 6 differ in product line evolution approach (infrastructure-based and branch-and-unite, respectively). Fig. 14 shows the comparison of the results with respect to time-to-market. The inputs specify that the evolution stage starts from Product 7. For Product 7, the

branch-and-unite approach has lower time-to-market because it can start product development without waiting for core asset update. For the later products, the branch-and-unite approach results in longer time-to-market because new product development can not be started until the previous product development has been finished and the core assets have been updated. This reduces concurrency, and explains the reason that the average resource usage for the branch-and-unite approach is less than the infrastructure-based approach, as shown in Fig. 15. The results also indicate that the branch-and-unite approach results in more effort than the infrastructure-based approach, due to the extra rework required on the new products after the core updates.

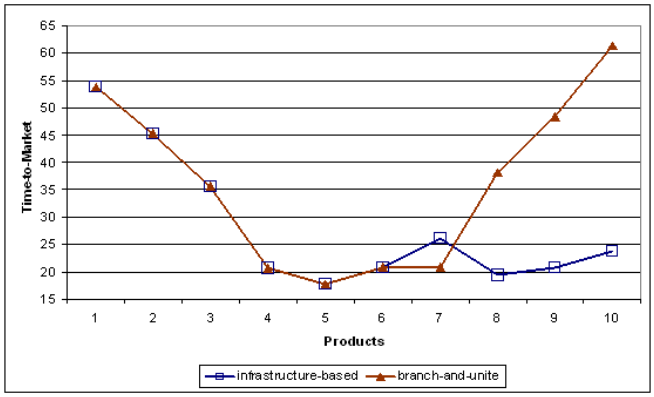


Fig. 14. Effect of evolution approaches on time-to-market

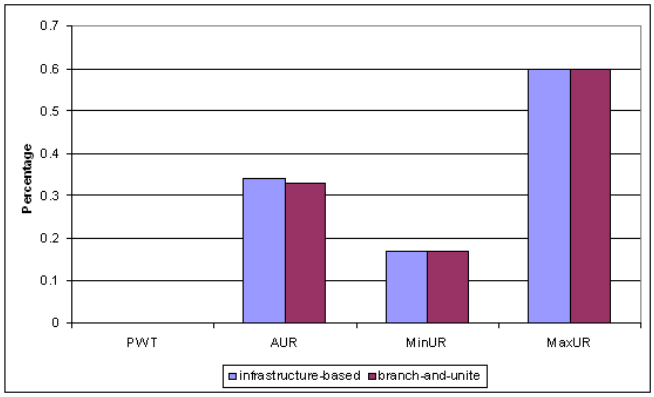


Fig. 15. Effect of evolution approaches on resource usage

D. *Effect of combined approaches*

Before developing a family of products, an organization needs to evaluate the available strategic options and determine which one best fits its goals. Strategies 4 - 8 show the alternatives an organization might have. Strategy 8 is the case for traditional software development (independent development) where products are created and evolved independently.

Fig. 16 shows the comparison of the alternatives in time-to-market. During the adoption stage (from Product 1 to 6), the time-to-market differences are mainly caused by the adoption approaches; during the evolution stage (from Product 7 to 10), the differences are mostly influenced by the evolution strategies. Because the transition from the adoption stage to the evolution phase happens between Product 6 and 7, the time-to-market transitions among different approaches occur between Products 6 and 7. As we can see, the big bang with infrastructure-based approach has the shortest average time-to-market, and the traditional approach has the longest average time-to-market. The traditional approach has the shortest time-to-market for the first two products, because it does not incur overhead from core asset creation. For the first three products, the big bang approaches have the highest time-to-market, because of the core asset development overhead. For Product 4, the incremental adoption has the longest time-to-market due to the core increase. For Product 6 through 10, the traditional approach results in the longest time-to-market, because of the higher development effort and lack of resources. By large-scale reuse, product line approaches generally result in smaller code size to development and maintain. Thus, the total effort and time on creating and evolving the products in a product line is smaller. Fig. 16, except the branch-and-unite cases, is consistent with the second hypothesis [14] proposed by Knauber *et al.*, which says that after some initial investment, the time-to-market per product decreases down to a certain minimum and significantly below the respective time-to-market in the case of independent development approach. For the branch-and-unite evolution strategies, although the time-to-market is still below the traditional approach, it has an increasing trend, which is caused by the dependencies imposed by this approach. This is consistent with Riva and Delrosso's observation from the product line development. They pointed out that some issues, such as organization bureaucracy and dependencies among tasks, can harm family evolution [15].

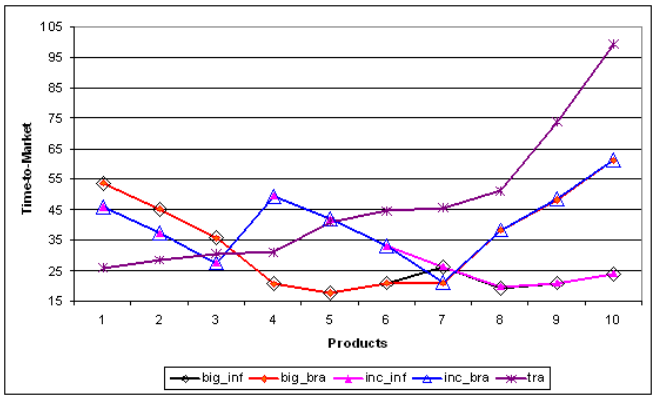


Fig. 16. Effect of combined approaches on time-to-market

Fig. 19 compares the product line initial development efforts (including initial product development effort and core asset development and increment effort) between big bang and incremental product line adoption and traditional approach. The big bang adoption involves highest initial development effort and lowest total effort. The traditional approach requires the lowest initial effort and highest total effort. The incremental adoption requires less initial effort but higher total effort than the big bang adoption. The figure suggests that there should be at least three products to make the product line approach appealing. These results are consistent with the product line investment curve described by Schmid and Verlage [7]. The results also confirms the first product line hypotheses [14] formulated by Knauber *et al.*, which states that after some initial investment, the effort that is needed to develop a new product decreases significantly below the effort that is required to build the

same product using independent development approach.

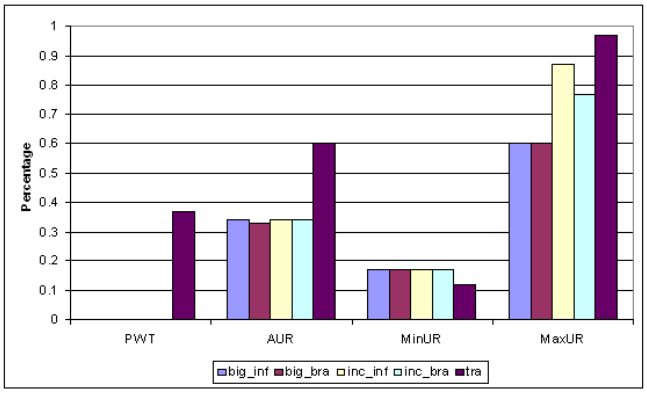


Fig. 17. Effect of combined approaches on resource usage

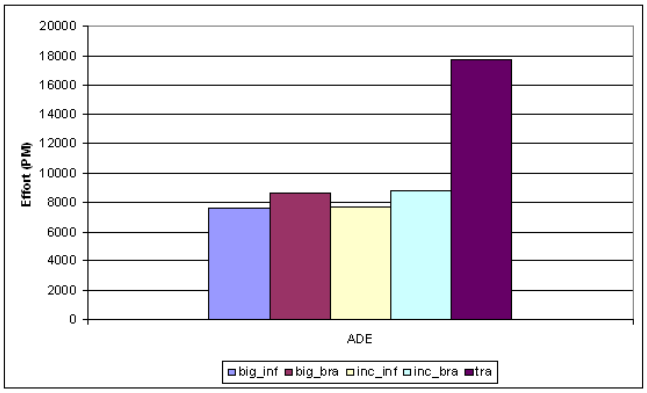


Fig. 18. Effect of combined approaches on accumulated development effort

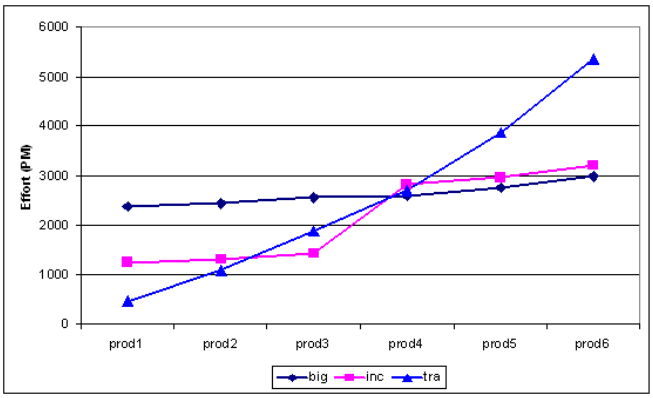


Fig. 19. Initial development effort comparison

V. Conclusions

Software product line approach requires higher up-front investment and results in increased process complexity. In order to reduce investment risk and ease process management, decision and process tools are necessary. In this paper, we described a simulator that is intended to support post-architecture software product line process impact estimation. It uses DEVSJAVA [4] as the modeling and simulation formalism, COPLIMO [5] as the cost model, and Microsoft Project [3] as the process definition tool. The simulator can provide both dynamic and static information for the selected process. Stepping through the simulator allows tracing product line process and uncovering hidden problems. The static results generated at the end of the simulation gives time, cost, and resource estimates for the selected process. By running different processes through the simulator and comparing their results, an organization can evaluate the alternative processes.

Sensitivity analysis has been conducted using the simulator. The results show that when reuse rate is high enough, resources are sufficient, and demands come at appropriate frequency, software product line processes result in lower life cycle costs, lower resource usage, and shorter average time-to-market. The results also show that the big bang adoption approach requires higher initial costs but lower total costs, compared to the incremental approach. In general, we feel the results of the simulator confirm common knowledge on software product lines [7]. In the future, we plan to solicit expert feedback and compare simulation results with real product line data.

In addition, we plan on incorporating other cost models into the simulator, using XPDL [16] as the process definition language, and combining optimization models with the simulator.

References

[1] P. Clements and L. M. Northrop, Software Product Lines: Practices and Patterns. Boston MA U.S.A.: Addison-Wesley, Aug 2001.

[2] M. Host, B. Regnell, J. N. O. Dag, and J. Nedstam, “Exploring bottlenecks in market-driven requirements management processes with discrete event simulation,” Systems and Software, vol. 59, pp. 323–332, 2001.

[3] Microsoft Corporation, “Microsoft project,” Jan 2005. [Online]. Available: http://office.microsoft.com/en-us/FX010857951033.aspx

[4] B. P. Zeigler and H. S. Sarjoughian, “Introduction to DEVS modeling & simulation with Java,” Aug 2003.

[5] B. Boehm, A. W. Brown, R. Madachy, and Y. Yang, “A software product line life cycle cost estimation model,” in ISESE ’04:The 2004 International Symposium on Empirical Software Engineering (ISESE’04). IEEE Computer Society, 2004, pp. 156–164.

[6] Y. Chen, G. C. Gannod, J. S. Collofello, and H. S. Sarjoughian, “Using simulation to facilitate the study of software product line evolution,” in 7th Intl. Workshop on Principles of Software Evolution, Kyoto, Japan, Sep 2004.

[7] K. Schmid and M. Verlage, “The economic impact of product line adoption and evolution,” IEEE Software, vol. 19, no. 4, pp. 50 – 57, Jul 2002.

[8] C. W. Krueger, “Easing the transition to software mass customization,” in the 4th International Workshop on Software Product-Family Engineering, Bilbao Spain, Oct 2001, pp. 282–293.

[9] J. Bosch, “Maturity and evolution in software product lines: Approaches, artefacts and organization,” in The Second Software Product Line Conference, San Diego, USA, Aug 2002, pp. 257–271.

[10] G. Bockle, P. Clements, J. D. McGregor, D. Muthig, and K. Schmid, “Calculating roi for software product lines,” IEEE Software, vol. 21, pp. 23–31, May/Jun 2003.

[11] S. Cohen, “Predicting when product line investment pays,” Software Engineering Institute, Tech. Rep. CMU/SEI-2003-TN-017, Jul 2003.

[12] B. W. Boehm, B. Clark, E. Horowitz, J. C. Westland, R. J. Madachy, and R. W. Selby, “Cost models for future software life cycle processes: COCOMO 2.0,” Annals of Software Engineering, vol. 1, pp. 57–94, 1995. [Online]. Available: citeseer.ist.psu.edu/boehm95cost.html

[13] K. Czarnecki and U. W. Eisenecker, Generative programming: methods, tools, and applications. Addison-Wesley, 2000.

[14] P. Knauber, J. Bermejo, G. Bockle, J. C. S. do Prado Leite, F. van der Linden, L. M. Northrop, M. Stark, and D. M. Weiss, “Quantifying product line benefits,” in PFE ’01: Revised Papers from the 4th Intl. Workshop on Software Product-Family Engineering. London, UK: Springer-Verlag, 2002, pp. 155–163.

[15] C. Riva and C. D. Rosso, “Experiences with software product family evolution,” in Sixth International Workshop on Principles of Software Evolution, Helsinki, Finland, Sep 2003.

[16] W. M. Coalition, “Workflow process definition interface - XML Process Definition Language,” Workflow Management Coalition, Tech. Rep. WFMC-TC-1025, Oct 2002.

Simulation Models Applied to Game-Based Training for Software Project Managers

Alexandre Ribeiro Dantas
COPPE UFRJ
Caixa Postal 68511
Rio de Janeiro, Brazil
alexrd@cos.ufrj.br

Márcio de Oliveira Barros
DIA UNIRIO
Av. Pasteur 458, Urca
Rio de Janeiro, Brazil
marcio.barros@uniriotec.br

Cláudia Maria Lima Werner
COPPE UFRJ
Caixa Postal 68511
Rio de Janeiro, Brazil
werner@cos.ufrj.br

Abstract

Inadequate use of project management techniques in software projects can be traced to the lack of efficient education strategies for managers. In this work, the learning-by-doing model is presented as an approach for project management education. This model requires an environment where students can act as managers without the risks associated to the failure of real software projects. The limitations of simulation tools and pilot-projects are discussed and a game-based training approach is proposed to address project management hands-on education requirements.

1. Introduction

Many studies have observed that the adoption of project management techniques in software development projects is usually deficient and inadequate [17][23]. Such studies suggest that these deficiencies may be a potential cause for so many projects experiencing poor quality, schedule and budget overruns. On the other hand, one possible trace to inadequate use of project management can be related to the lack of comprehension of its essential practices and techniques.

Since project management is accepted as a knowledge-intensive activity, personal expertise drives its major success factors. However, the common practice of promoting new managers from the technical staff, due to their success when applying technical skills, and the lack of efficient project management education strategies contribute to this scenario of inadequate management [17]. While novice managers usually take "fire-fighting" actions based on intuitive decisions which can lead to budget and schedule overruns, experienced managers apply lessons learned from past projects, even if these were unsuccessful ones (the learning-by-error approach) [13].

By analyzing past situations and evaluating the different paths that the project could have followed if specific decisions were made, a student can enhance his management skills. In this sense, case studies analysis and role-play strategies are useful to develop the students' expertise. This proper investment in project management education, emphasizing a hands-on learning approach, seems to play an important role in preventing future faulty projects due to inadequate management.

In this work we propose a simulation-based game as an educational instrument to provide learning-by-doing for project managers. A System Dynamics [10] software project model describes how the game will react to user decisions. A simulator and a game machine complement the game. While the simulator is responsible for executing model equations, the game machine handles user interactions. Such interactions are inserted into the simulation and the game machine provides feedback results in a game-like fashion, through clear goals, challenge, drama and multimedia effects.

The remainder of this paper is organized as follow. Section 2 discusses the challenges to educational models for project management, emphasizing the learning-by-doing approach. Section 3 presents our approach to add game features to simulation models. Section 4 discusses the limitations of the modeling approach that we have selected and the changes that were made to introduce game-related functionalities into this simulator. Section 5 provides qualitative evaluations about the use of a game-based training model while section 6 presents our final considerations.

2. Towards an experience-based learning approach for project management

The pedagogical field, the art and the science of teaching present diverse education models. Such models were developed for specific contexts, ages, perception and memory styles. They describe the

main interactions among the instructor, students, supporting materials and teaching resources. In order to analyze which education model would best fit project management characteristics and requirements, we address some pedagogical models deficiencies.

Most current education models are content-driven and instructor-centric: a teacher is responsible for selecting what students will learn, when and how the learning process will be conducted. These models are usually supported by expositive classes, seminars, textbooks and tests [20]. Some studies have shown that this approach is not adequate for adult learning [13], since adults prefer to learn based on experience. They must be motivated and learn better what they can apply to solve their current problems. Management education is strictly adult learning and motivation is a strong factor for the educational success. Experiencing faulty projects due to insufficient or inadequate management, for example, is one way of increasing adult's motivation and engagement to an artificial learning situation [8].

In addition, large-scale software projects have complex nature. This nature is characterized by dynamic complexity in feedback loops, non-linear behavior, cause-and-effect relationships distant in time and social influences [24]. Complex project behavior usually cannot be estimated with precision by human perception, intuition and mental models. On the other hand, by living situations that require specific decisions and analyzing the behavior derived from choices previously made, the student builds a library of behavioral patterns relating decisions and their consequences on software projects. Later, this knowledge will be the basis of students' managerial decisions. While analyzing a decision, a manager usually seeks in his memory for a similar situation in other projects or uses his perception to capture current reality and mentally predict its future state according to the available alternatives [7].

Since software project behavior is not easily derived from basic principles (the content to be learned), content-driven educational approaches should be complemented with techniques that exploit learning-by-doing. Pilot-projects and simulation are good examples. The use of simulations benefits from reduced schedule, budget and risks constraints inherent to real and even pilot projects. Moreover, a simulation model can be configured to represent distinct development situations that could only happen in real large projects, with long schedules and large teams. In this case, different models can be quickly executed and analyzed by simulation with a variety of educational, process evaluation or improvement goals.

Thus, learning-by-doing is a strong requirement for adult education models and a key issue for a better project management educational approach. However, a simulation is not necessarily a learning environment. There is still a lack of foundation in cognitive learning theory and a coupling of that theoretical perspective with the practice of instructional design [22].

The efficiency of simulation use in education is yet to be well documented and established [11][22]. When conducting research on the effectiveness of business simulators, issues beyond the simulator should be considered, including the user characteristics and the learning situation. There are actually many subjective and intervenient variables that can lead to biased empirical results, especially when comparing different approaches such as books, teachers and simulators (e.g., teacher's quality, book's content). Similarities of approaches and complementarities among them should be exploited, too [12].

Business simulators (also known as 'business flight simulators') are tools for an improved experience-based decision support. There are many examples of those flight-simulators in the industry and technical literature [6][12][16][18][19][26]. Usually, those simulators present similar characteristics: a business process model, a control panel to initial conditions setup and key management decisions (using input boxes, knobs, dials, etc), text reports and graphs. Some empirical studies [16][18] have addressed the use of simulation and modeling in an educational context. These studies indicate that a simulation model approach with role-play scenarios is considered to be useful, providing understanding about software projects behavior in project management learning.

Two empirical studies were recently conducted [2] to evaluate the use of simulation to support decision-making on software project management. Such studies illustrate some simulation drawbacks for educational goals. Students from two different universities (3 D.Sc. students, 26 M.Sc. students, 16 B.Sc. students and 4 B.Sc.) were asked to manage a small project with a major objective of concluding it in the lowest schedule as possible, while attending to certain quality restrictions.

A project emulator (that is, a software that dictates project behavior over time) was used to represent the proposed project. The participants interacted with the emulator, making decisions about which developers should take part in the project team, how many hours should each developer work per day, if inspections should be included in the development process, and which developer should accomplish each project activity. Half of the participants managed the project based on their own experience. The remaining participants used simulations to analyze their options and evaluate

their decisions before applying them in the project emulator.

The results of these studies showed positive correlation between subject experience, ability to interpret simulation results and success in attending to project objectives. This was an unexpected result because modeling and simulation were supposed to provide more help to inexperienced managers. It was also observed that the lack of engagement had negative influence on subjects' performance.

In order to have an engaged learning-by-doing experience, the "plan the flight and fly the plan" approach usually found in business flight-simulators may not be sufficient. A more active student participation and control during the simulation should be addressed instead of just setting up initial model conditions and passively watching simulation results. In an artificial learning situation, it is usually difficult for a model analyst to trace model observed results to intermediate behavior. A graphical feedback of simulation results can perform better for project behavior understanding and cause-effect mapping, especially to novice managers. Simulation tools lack the look-and-feel of a real project development environment. To make modeling and simulation more useful for inexperienced managers, we shall look for better ways to present simulation results.

A hands-on adult learning model for project management requires an environment where students can act as managers. In this sense, games can be integrated to simulation models, adding fun, challenge, visual effects and a more compelling interaction model for students. Interaction is one of the fundamental game characteristics for an active learning construction and to avoid player's boredom in the activity. With an active and flexible participation, the player tastes the control of the game, exploring its contents and seeking defined goals, following his own way to overcome the challenges. Usability, fidelity, multimedia feedback and drama effects are also important issues to provide the role-play experience to the player, acting as a manager in a virtual software development office.

Although playing is still considered to be the opposite of working for many people and organizations, digital games are increasingly being adopted in serious-minded applications for experience-based therapy, communication and education [25]. The computer and video games software sales in the United States grew four percent in 2004 to U$ 7.3 billion [9]. Digital games are a growing market to children and also to adults: the average age of a game player is 36 years [9] while the average task-force age is 39 years [20]. Thus, soon all the task-force will consist of people who have always lived in a digital world with strong participation of games. According to the Entertainment Software Association, more than half of game players expect to be playing as much, or more, as they do today 10 years from now [9].

The game-based educational approach can have multiple targets such as individual and organizational learning, prevision and decision-making support, process evaluation and improvement. Similar approaches for software engineering education can be found in the technical literature, such as the *SimSE* tool [20], the *SESAM* project [7] and for military training, such as the *Full Spectrum Command* [15].

Finally, an education game approach requires some investment in order to be built and prepared for application in classes. In order to enhance the results of such investment, a game machine should be reusable across several distinct learning experiences. In this work, we emphasize some important requirements for a reusable education simulation-based game: simulation model flexibility through reusable knowledge pieces, graphical user interaction and feedback through multimedia features. By introducing the model as a separated component using pluggable pieces of knowledge, different educational goals and situations can be performed by trainees reusing the same game machine and simulator.

3. Simulation models for an education game machine

To evaluate the game-based learning approach, we developed a simulation-based game, called *The Incredible Manager*[1]. The game machine allows a trainee to act as a project manager, being responsible for planning, executing and controlling a software project. The trainee's goal is to complete a proposed project within the cost and schedule established during a planning phase and approved by the project's stakeholders. While planning and controlling the software project, the trainee has to make several decisions, including team selection, allocation, effort dedicated to quality assurance, budget and schedule control, among others.

Virtual characters, whose behavior is controlled by a simulation model, represent project stakeholders. They inspect the trainee's project plan, comparing it to a baseline plan based on the average cost and effort required to build the project. The stakeholders must approve the trainee's plan in order to authorize project execution. Such approval allows the trainee to review a plan requiring much less resources than should be expected to complete the

[1] The game and other System Dynamics resources are available at <http://reuse.cos.ufrj.br/riosim/>

project, but it also inhibits the trainee to prepare a plan requiring much more resources than needed to develop the project.

After having an approved project plan, the trainee can start the project development. The development runs in continuous turns, consuming project resources requested by the project plan. The player must be aware of the project behavior and take corrective actions when necessary. Visual effects and reports show the game characters' state, such as exhausted developers, late tasks, project without funds, and so on. To avoid finishing the resources before project completion, the player may need to modify the original plan on the fly. According to these decisions, different players can live the experience of managing the same simulation model of a project in different ways.

The simulation model describes the world and relevant aspects of the software project presented to the trainee. However, software development projects are difficult to model since they are classified as systems of complex dynamics [24]. Addressing these difficulties, System Dynamics [10] is a modeling discipline based on a holistic view to describe and evaluate the visible behavior presented by a system. Such behavior is determined by the structure of elements that participate in the system and the relationships among them, described in the model through mathematical equations.

This modeling discipline has already been used in the development of software project models [1], which became a base for subsequent reviews and extensions by other authors. One of these extensions is the scenario-based project management paradigm [3][4], which separates uncertain aspects from known facts in project models. This separation occurs by building distinct models (namely scenario models) for each uncertain aspect that can influence a software project, while the project itself is described in a baseline model (or domain model) to which the separate models will be integrated. Since project behavior can be affected by unexpected events, management actions and strategies, the student may test its sensibility to combinations of such elements, evaluating their impact over the project success.

A proposed System Dynamics metamodel [4] encapsulates the complexity of the traditional constructors in an object-oriented fashion. Models can be easily developed, modified, integrated and expanded to embrace management knowledge from the technical literature and practice. The metamodel is the basis for building domain and scenario models. A domain model defines categories of elements that take part in the problem domain represented in the model, their properties, behavior and relationships. The domain model does not describe a model for a specific problem, but for a knowledge area where modeling can be applied. It is a domain description that must be specialized to describe a particular problem. Thus, from an abstract domain model it is possible to create several operational project models.

Scenario models provide a library of generic management events and theories that an instructor can integrate to a project model and present to management trainees during a simulation session. Senior managers can develop scenario models expressing experiences they have collected by participating in several projects. These scenarios will further allow less experienced managers to share senior managers' knowledge. We have a library of about fifteen scenario models available for the simulator. These scenarios include theories regarding developers' productivity and error generation rates due to their experience, developers' productivity due to learning the application domain, effects of overworking and exhaustion upon developers, communication overhead, error propagation along the activities that compose a project network, among others.

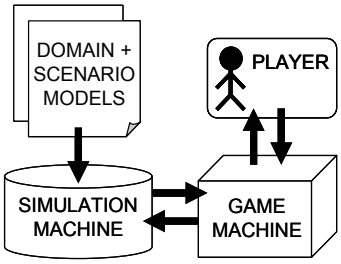


Figure 1. Game components and interactions

Main game components and their interactions are shown in Figure 1. The simulation model (composed by a domain model and selected scenario models) is an independent component that can be easily exchanged according to the educational goals or management knowledge desired. In fact, each simulation model represents a game phase (externally set in a configuration file). Phases are independent files that can be used to create multiple educational projects with increasing complexity or peculiarities.

The high-level models (domain and scenarios) can then be translated to traditional System Dynamics constructors for computational simulation. The System Dynamics metamodel translation and simulation process, however, must present especial features considering the requirements for a game-based simulation. These features are presented and discussed in the next section.

The game machine differs from a traditional simulator display by presenting model behavior and its changes over time in a multimedia format that tends to be easy to understand by the students. Moreover, their interaction with the simulation

process is made through an interface closer to the real-world situation of managing a software project: instead of handling with mathematical equations, the trainee deals with iconographic representations of developers, activities, and other elements that are usually present in a real software project. As shown in the qualitative evaluations that we conducted, this enforces motivation and, therefore, enhances the learning process.

Figure 2 shows the game office environment. Color changes indicate the project network (in the top right), schedule and funds (in the bottom left) current states. Developers' states are shown through balloons and body expressions as they are idle, tired or in panic. The player can pause the game, ask to modify the project plan and visualize selected developers and task details in the message area (in the bottom right).

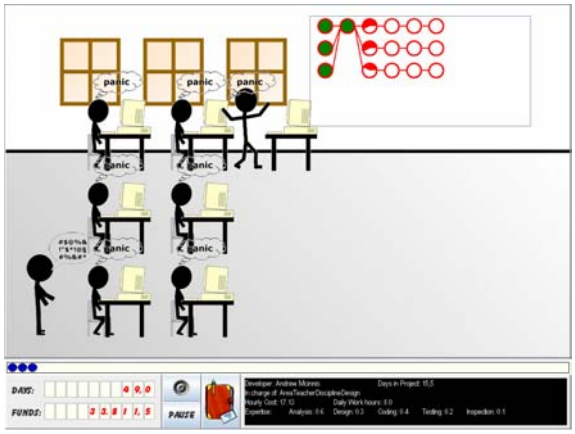


Figure 2 - The game office

4. Dynamic-structure simulation

Most traditional System Dynamics simulators provide a static-structure simulation: although the behavior of domain elements changes over time, their structure, that is, the relationships among these elements, remain fixed from the first to the last simulation step. Usually, model properties are initially configured by an operator and the model behavior is evaluated after all simulation steps are executed.

This type of simulation is not well suited for game-base simulations, where user interaction is a fundamental characteristic. Such interactions describe player actions to control the project. Like in the real world, the manager should be able to hire new team mates, ask them to work overtime due to schedule pressure or increase the effort on design inspections to not impact the product quality. In a project management game, these user decisions may require changing the way in which domain elements relate to each other. For instance, we consider that the developer selected to accomplish an activity is a structural relationship; since the manager can change the developer in charge of an activity (for many reasons, from the availability of a more capable developer to a request of the developer itself), the model structure should be able to change during the game session, without loss of previous simulated values during the previous simulation steps.

To address these limitations, this work presents a dynamic-structure compiler and simulator for models based on the System Dynamics metamodel [3]. The simulator starts by translating the project and scenario models built according to the System Dynamics metamodel to traditional System Dynamics constructors (this process is handled by the original metamodel compiler) [4][5]. Once the models are translated to mathematical equations, simulation steps are executed continuously until the player performs an action. Upon receiving a player action, the simulator performs the required structural changes in the high-level project model representation and translates the new model to the traditional equations. A player action may trigger the inclusion of new elements to the project model, removal of existing elements, or modifications upon the values of element properties. Once the structural changes are applied and the new equations-based model is generated, model simulation is restarted in the next simulation steps with the new set of equations, however without affecting the simulation steps already accomplished and the values calculated in these steps.

5. Qualitative evaluation

To evaluate the game-based approach for project management education, 24 people from 3 different groups were invited to run *The Incredible Manager* with a simulation model representing a software project. The first group was composed by 7 participants from a project management graduation class. The second was composed by 8 participants from an industrial software development laboratory, while the third was composed by 9 participants (organized in 3 groups) from an undergraduate class. A total of 11 B.Sc. students and 13 M.Sc./D.Sc. students ran the game.

While performing the study, participants were asked to play the role of a trainee using our game. The training session included a simulation (running the game) and a discussion session (presenting the lessons learned). All participants were asked to fill in two questionnaires. The first was filled in before the training section and contained questions about academic degree, personal experience and interest in software development and project management. The second questionnaire, filled in at the end of the training session, contained qualitative questions

about the game-based educational approach and the game itself.

The results of the qualitative questions for the three different evaluations were very similar: 100% approved the game-based model, felt that they learned the lessons presented, and that they increased their management skills. For 52.2% of the participants, the training session was considered very pleasant. Finally, 87.5% of the participants described that the game experience raised their interest in project management.

The remaining questionnaire results indicate that the game-based simulation learning is motivating, practical and fun for the participants. Challenge, visual effects and time pressure are also viewed as important factors for the engagement and entertainment during the activity. The participants, especially the novices, pointed out that the graphical feedback and the possibility of practical simulation of real project situations were very stimulating. The diversity of choices and the uncertainty from the model give a non-linear experience running the game in different situations, exploring several educational goals.

The most reported improvement points are related to the simulation model evolution, including new project development processes, psychological, social and organizational aspects of uncertainty. Multi-player support with different roles, interactions and an intelligent integrated coach during game execution are requested as future works.

The positive results achieved in the evaluation, though, cannot support the effectiveness of the game-based project management education. The focus of this evaluation was mainly to gain qualitative insights and future work directions. As stated before, the design and execution of experimental and quantitative studies using simulations in education is a complex task due to several intervenient variables to consider when isolating and measuring the effectiveness of such approach.

6. Conclusions

Simulation-based games seem well suited to be introduced in a learning-by-doing education model, such as required by manager trainees. They give the students an opportunity to experiment the consequences of executing or neglecting important project management functions, confront themselves with complex issues that must be resolved during project development, and test different approaches and solutions to project management problems. The lessons learned from the decision-making and the cause-effect discussions over their own projects become an important factor to improve and maintain the motivation and engagement necessary for a better artificial management training situation.

Concerning future works over the proposed models for game-based simulation, there is a special demand for graphical tools for model construction and evolution. Model refinements and new scenarios should include more realistic situations of software development offices, projects and developers. Since there must be a quantitative basis to these model evolutions, new empirical studies should be performed to gather these data.

Future research on software engineering game-based learning works should involve multiple disciplines, such as pedagogical issues over the training environment and process, cognitive and motivational issues, including usability, interaction, perception and multimedia presentation of the game.

Acknowledgments

The authors would like to thank all the participants involved in the studies and CAPES for the financial investment in this work.

References

[1] Abdel-Hamid, T.K., Madnick, S.E., *Software Project Dynamics - An Integrated Approach,* Prentice-Hall, Englewood Cliffs, 1991.

[2] Barros, M.O., Werner, C.M.L., Travassos, G.H., "Supporting Risk Analysis on Software Projects". *The Journal of Systems and Software,* v. 70, n. 1-2, 2004, pp. 21-35.

[3] Barros, M. O.; Werner, C.M.L.; Travassos, G. H., "System Dynamics Extension Modules for Software Process Modeling". In: *International Workshop on Software Process Simulation and Modeling*, Portland, 2003.

[4] Barros, M.O., Werner, C.M.L., Travassos, G.H., "A system dynamics metamodel for software process modeling". *Software Process Improvement and Practice*, v.7, n.3-4, 2002.

[5] Barros M.O., "Hector – System Dynamics Metamodel Compiler". Available at <http://sety.cos.ufrj.br/riosim/tools/Hector/Hector.php>, March 25, 2005.

[6] Beer Game Resources, Available at <http://web.mit.edu/jsterman/www/> March 25, 2005.

[7] Doyle, J.K., Ford, D.N., Radzicki, M.J., Tress, W.S., "Mental Models of Dynamic Systems". Available at <http://www.wpi.edu/Academics/Depts/SSPS/Research/Papers/27.pdf>, March 25, 2005.

[8] Drappa, A., Ludewig, J., "Simulation in Software Engineering Training". In: *Proceedings of the*

International Conference on Software Engineering, Limerick, 2000, pp. 199-208.

[9] Entertainment Software Association. Available at http://www.theesa.com/facts/top_10_facts.php, March 25, 2005.

[10] Forrester, J.W., *Industrial Dynamics,* Cambridge, MA: The MIT Press, 1961.

[11] Größler, A., "Methodological Issues of Using Business Simulators in Teaching and Research". In: *Proceedings of the Conference of the International System Dynamics Society*, Bergen, Norway, 2000.

[12] Größler, A., Notzon, I., Shehzad, A., "Constructing an Interactive Learning Environment (ILE) to Conduct Evaluation Experiments". In: *Proceedings of the Conference of the International System Dynamics Society*, Wellington, New Zeland, 1999.

[13] Klein, G., *Sources of Power: How People Make Decisions*, Cambridge, MA: MIT Press, 1998.

[14] Knowles, M., *Andragogy in Action*, Jossey-Bass, San Francisco, CA, 1984.

[15] Macedonia, M. Ender's Game Redux. IEEE Computer, v.2, 2005, pp. 95-97.

[16] Maier, F.H, Strohhecker, J., "Do Management Flight Simulators Really Enhance Decision Effectiveness". In: *Proceedings of the Conference of the International System Dynamics Society*, Cambridge, 1996.

[17] Mandl-Striegnitz, P.; Lichter, H., "A Case Study on Project Management in Industry: Experiences and Conclusions". In: *Proceedings of the European Software Measurement Conference*, Antwerp, 1998, pp. 305-313.

[18] Pfahl, D., Laitenberger, O., Dorsch, J., Ruhe, G., "An Externally Replicated Experiment for Evaluating the Learning Effectiveness of Using Simulations in Software Project Management Education". *Empirical Software Engineering*, 8, 2003, pp. 67–395.

[19] POWERSIM Resources, Available at <http://www.powersim.com/> March 25, 2005.

[20] Prensky, M., *Digital Game-Based Learning*, McGraw-Hill, 2001.

[21] SimSE Online Resources, Available at http://www.ics.uci.edu/~emilyo/SimSE/, March 25, 2005.

[22] Spector, J.M., Davidsen, P.I., "Constructing Learning Environments Using System Dynamics". *Journal of Courseware Engineering*, v. 1, 1998, pp. 5-11.

[23] Standish Group, "The Chaos Chronicles", The Standish Group International, 2003.

[24] Sterman, J.D., *System Dynamics Modeling for Project Management*, MIT System Dynamics Group, Cambridge, MA, USA, 1992.

[25] Swartout, W., van Lent, M., "Making a Game of System Design". *Communications of the ACM*, v.46, n. 7, 2003, pp. 32-39.

[26] VENSIM Resources, Available at <http://www.vensim.com/>, March 25, 2005.

Towards Interactive Systems Usability Improvement through Simulation Modeling

Nuria Hurtado[1], Mercedes Ruiz[1], Jesús Torres[2]
[1]*Department of Computer Languages and Systems*
University of Cadiz (Spain)
{nuria.hurtado, mercedes.ruiz}@uca.es
[2]*Department of Computer Languages and Systems*
University of Seville (Spain)
jtorres@lsi.us.es

Abstract

Nowadays, usability has become an essential contribution to the success of interactive systems and is recognized as a quality attribute for software products. This paper proposes the use of dynamic simulation models for the improvement of interactive systems usability through the application of a User Centered Design (UCD) process and its integration into the software development process. The simulation model developed is used to experiment on the effect that different levels of usability have over the behavior of the UCD process in a specific kind of interactive systems such as web site application development.

1. Introduction

Over the last few years, there has been an increase in the amount of people using and depending on computer technology. At the same time, due to the growth and expansion of the internet, software systems have increased their interaction degree. This implies an ever-growing demand of more usable products.

For a long time, the importance of usability has been neglected in the development of interactive systems, and so it has been relegated to nothing else than final product evaluation activities. It is important to bear in mind that system usability is not only related to user interface appearance but, mainly, to the way in which the user can interact with the system and, hence, to the overall structure of the system and the logic of the business.

Usability increases customer satisfaction and productivity, leads to customer trust and inevitably results in tangible cost, savings and profitability [15]. Thus, the software industry should realize that they need to pay attention to usability from the early stages of system development with the introduction of a User Centered Design (UCD) approach.

Along these lines, different proposals have been made, coming from both the Usability Engineering (UE) and the Software Engineering (SE) fields, for the setting out of methods, techniques and tools with the aim of orienting developers as to which activities should be carried out during the software development process that may grant a previously established usability level [2][5][6][7][11][16].

However, in spite of the social and economic benefits that usability allows and yet despite strong motivation within some organizations to practice and apply effective SE and UE methods, there still exist major gaps of understanding both between suggested practice, and how software is actually developed in industry, and between the best practices of each of the fields. The existing UE methods are integrated in development practices in a way that is more opportunistic than systematic. As a result, product quality is not as high as it could be, and rework is often necessary [9].

Modeling and simulation techniques are considered as valuable tools for the improvement of processes in several areas of engineering. Since the early 90s various simulation models have been developed to respond to different questions related to the software development process proving their usefulness in this scope [13].

This paper presents an approach to the application of modeling and simulation techniques to the User Centered Design (UCD) process and usability improvement. More precisely, it proposes the use of dynamic simulation models for the improvement of

interactive systems usability through the application of a UCD process and its integration into the software development process [10].

The proposed approach is intended to help developers understand and improve the behavior of the UCD process and its special features, reinforcing motivation for a change in the development process of organizations, and helping to bridge the existing gaps between SE and UE.

For the purpose of this study, the simulation model is used to determine the effect that different levels of usability have over the UCD process behavior of a specific kind of interactive systems development such as web site design.

The structure of the paper is as follows. Section 2, presents the concepts of usability and UCD in order to set the scope of our study and we comment on the process model that is eventually chosen to build a simulation model. Section 3, presents a brief account of the advantages of simulation models of software processes that support the usefulness of the application of these techniques to the UCD process. Section 4, introduces the model development, as well as the chosen simulation approach, a description of the model and parts of it, the definition of scenarios for the simulation and some simulation results. Section 5, includes the main conclusions of this proposal and future work to be carried out along these lines.

2. Usability and User Centered Design

The term usability is defined in norm ISO 9241-11 as "the degree to which a product may be used by any given users to attain specific objectives with effectiveness, efficiency and satisfaction in a specified context of use" [12].

It is necessary to point out that usability depends strictly on the context of use, that is, on specific users and work environment, and hence it is a quality not inherent to software. Hence, it is deduced that in order to develop a usable product it is not enough to systematically apply any general instructions or usability guidelines, but it is necessary to apply a UCD process that allows for the integration of the user into the development from the early stages of it, thus permitting an extensive knowledge of the context of use.

User Centred Design is an approach to interactive systems design that specifically aims at making systems more usable through the incorporation of the user to the development process.

Amongst the benefits of the application of UCD processes the ISO 13407:1999 [11] includes:

- Cost production reduction. Cost and development time can be reduced, avoiding redesign and reducing the number of later changes on the product.
- Increase of user productivity and operational efficiency of organizations.
- Improvement of the quality of the product and of its appeal to users, resulting in a competitive advantage.
- Making systems that are easier to use and learn, thus reducing the cost of technical service, training and maintenance.
- Increase of user satisfaction, which reduces trouble and stress.

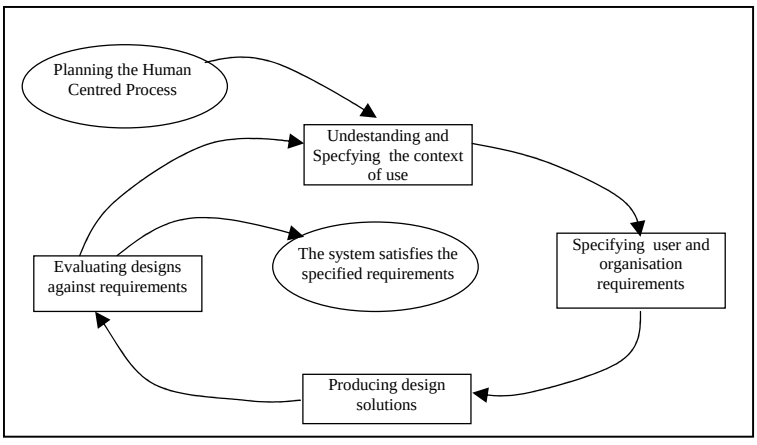


Figure 1. Interdependence of User Centred Design activities [11]

2.1. Model of User Centered Process

As it has been already mentioned, there are different methodological suggestions, coming from various disciplines (UE and SE) for the development of interactive systems based upon the user centered approach. All these proposals aim at guiding developers in proceeding in an organized way in order to attain the usability of an interactive system during its development, although how the integration of HCI (Human-Computer Interaction) proposals into SE Process Models should be carried out, is still under research. The present work is centered in the process model developed in the international standard ISO 13407:1999 [11] since it is considered to be the basic reference in the development of user centered processes by the HCI community. It is not linked to any existent methods, and it complements any design methods and lays down a user centered general perspective that may be integrated into various development processes according to each particular context. All design activities introduced are applicable, to a greater or lesser extent, to each of the system development stages, although -previous to its application- a user centered planning of the process must be set up. Such planning must include, among other things, the procedure for the integration of these activities into the rest of the system development activities (for example analysis, design and evaluation). Such procedure will depend in each case on the project in particular but it should always allow for iteration. Nevertheless, the standard does not specify how such integration must be done. Figure 1 shows the various activities of the UCD process and interdependence among them.

The process describes four main design activities centred on the user: understanding and specifying the context of use, specifying user and organization requirements, producing design solutions and evaluating design against requirements. The process implies the iteration of these activities until the system satisfies the specified requirements. A brief explanation of each activity follows:

- Understanding and specifying the context of use. Identification should be made of the features of potential users, the tasks they are going to perform and the environment in which the system is going to be used.
- Specifying user and organization requirements with respect to use context description. Objectives must be set identifying compromises and priorities among the various requirements.
- Producing design solutions. Specific design solutions must be carried out using some kind of prototyping. Such prototypes are presented to users and feedback is used to make design modifications.
- Evaluating design with respect to requirements. Evaluation must be present at all stages of the life cycle, with the intention of providing a feedback that contributes to design improvement. It will also determine whether the specified objectives have been attained, and it will check the use of the product in the long term.

3. Modeling and Simulation for Software Process Improvement

Simulation can help when it comes to make decisions about questions related to process improvement, because it helps predicting the effect that a change would have in the process before it takes place.

In this scope, the dynamic model introduced in [14] is of great importance, being – along with Abdel-Hamid´s original model [1] - one of the dynamic models that represents with greater detail the whole software development process. In [14], a model for showing the effect of making formal inspections on cost, deadline and quality of projects is introduced. Also, the use of simulation models to predict, quantitatively, the impact of changes upon processes is proposed in [20].

Most recent simulation models are especially designed and oriented towards the evaluation of the results of different measures for process improvement. Various models have been developed in the scope of the CMM model (Capability Maturity Model) among which, the models proposed in [21] and [22] are worth pointing out. [21] shows the application of a model to predict software process perform in terms of effort, staff, deadline and quality of the product. A Dynamic Integrated Framework for Software Process Improvement (DIFSPI) is developed in [22]. It offers a methodology and a working environment that combines both the advantages of traditional methods and those of systems dynamics, thus allowing project managers as well as members of the Software Engineering Improvement Group (SEIG) to design and evaluate new software process improvements.

[10] presents an initial approach to the application of modeling and simulation techniques to the UCD process.

4. UCD Process Modeling and Simulation

4.1. Simulation approach

There are several simulation model approaches applicable to the study of the various aspects of the software process. Among them, two main approaches are worth pointing out: Continuous modeling and discrete modeling.

The continuous simulation approach is based upon the Systems Dynamics theory. It is useful when systems contain variables that change in a continuous manner with time. Continuous modeling of a process represents the interaction among its key factors as an ensemble of differential equations where time is increased step by step.

The discrete simulation approach is based upon queue systems. In the discrete simulation, time advances when a discrete event takes place.

Since the purpose of this study is to model UCD process mechanisms, we have chosen the continuous simulation approach.

4.2. Model Development

4.2. 1. Introduction to model development

The main aim of the developed model is to help understand and improve the UCD process and its integration into the overall software development process, resulting in the improvement of system usability. The process model established in ISO 13407:1999 [11], has been chosen to model and simulate the UCD process. The computation of the amount of tasks to be developed in each activity of the UCD process, as well as the effort to be allocated to each of them, have been adapted to the special scenario of a web site design project [3].

4.2.2. Estimations of Usability Effort and Usability Size

For the estimation of Usability Effort and Usability Size, several input parameters and auxiliary variables have been considered. Figure 2 shows the diagram for the computation of these variables.

The input parameters involved are the following:

- `Web Project Size`: measured in thousands of Source Lines of Code (SLOC)
- `Life Cycle Phase`: This parameter determines the development stage of the project that is going to be simulated, namely, early, central or late stages. These stages could correspond to the analysis, design and evaluation phases of a classic development life cycle.
- `Usability Level`: This parameter determines the level of usability of the project. The parameter can take three different values, low level, medium level and high level.

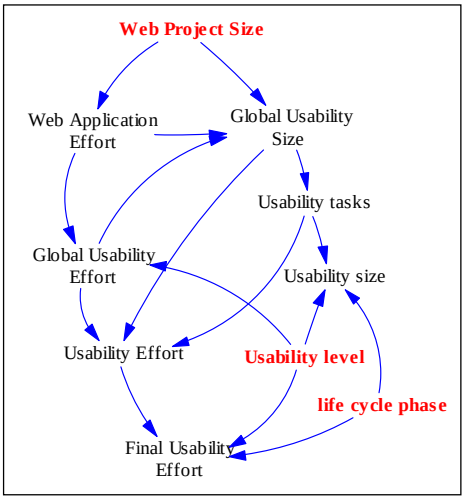


Figure 2. Usability effort and usability size estimations

The variables involved are the following:

- `Web application effort` measured in person_month has been estimated using the estimation model called WEBMO [18][19]. This model estimates the effort and duration of web applications development projects as an adaptation of the COCOMO II early design model [4]. The effort equation used is the following:

$$Web\ application\ effort = A\prod_{i=1}^{9} cd_i\,(Web\ \ project\ \ size)^{P1}$$

The constant *A* and the values for the power law *p1* will depend on which of the five application domains is considered. The application domain considered in this model is the one that corresponds to the web portals domain. The equation has also nine cost drivers *cd*, which have been set to their nominal values in this case study.

- `Global usability effort`: This value is obtained using `Web application effort` according to the conclusions of the research carried out by Nielsen Norman Group in their study of the best usability practices in web development [17]. For the purpose of this model, the obtained value has been considered as the nominal value corresponding to a medium

usability level (level 2). The value will be increased or decreased by a percentage law for the usability levels: high (level 3) and low (level 1), respectively.

- `Global usability size`: measured in thousands of SLOCs.
- `Usability tasks`: measured in tasks.
- `Usability effort`: Global effort for the usability tasks.

Finally, the values corresponding to the `Usability size` and the `Final usability effort` variables are obtained depending on the `Usability level` and the `Life cycle phase` input parameters, conforming to the proposed scenario.

4.2.3. UCD Process Modeling

Figure 3 shows a simplified flow and level diagram of the developed model. Each of the activities of the UCD process described in [11] has been represented as a level variable. Level variables represent the number of tasks that are performed on each of UCD activities, namely:

- `Specified context of use.`
- `Specified user´s requirements.`
- `Designed solutions.`
- `Evaluated design solutions.`

The percentage of `Usability Size` variable that it is necessary to perform on each activity will depend on the `Life cycle phase` input parameter. According to such parameter the initial values for the various UCD activities will vary. These initial values are represented by variables:

- `Initial size of context of use.`
- `Initial Requirements size.`
- `Initial Design size.`
- `Initial Evaluation size.`

In order to control the sequence for the activation of each activity the model is based on the following pattern: When a certain percentage of tasks is completed on a particular activity, it will be possible to start the next activity. This percentage is established through a series of input parameters. Each of these input parameters act upon the following auxiliary variables that control the start of activities:

- `Necessary context of use.`
- `Necessary user´s requirements.`
- `Necessary solutions.`
- `Necessary evaluation.`

Each activity gets started by the activation of the corresponding flows. The flow is activated when the number of performed tasks satisfies the percentage of tasks established as necessary to be able to go on to the next activity.

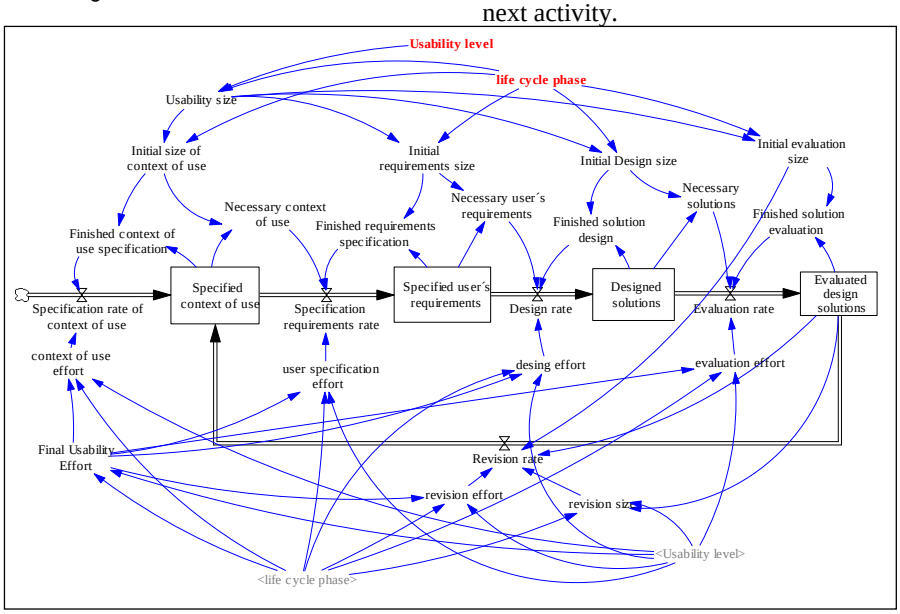


Figure 3. Simplified flow and level diagram

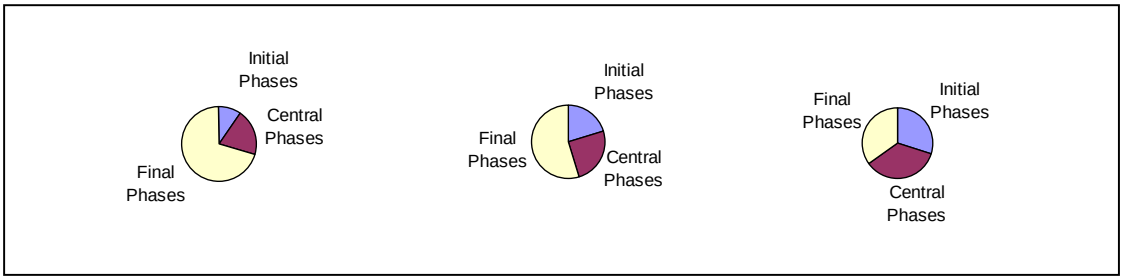


Figure 4. Usability Tasks Global Distribution in the life cycle according to Usability Level

The flow of work, flows applying a development rate between one activity and the next one. The development rate will depend on the productivity and dedication of the staff assigned to each one of the UCD activities as well as on the effort corresponding to each activity. Flow variables are as follows:

- `Specification rate of context of use.`
- `Specification requirements rate.`
- `Design rate.`
- `Evaluation rate.`
- `Revision rate`: This rate will be affected by the `revision size` variable that will agree with the percentage of evaluated tasks that need to be re-elaborated and will depend on `Usability level` and `Life cycle phase` input parameters.

The final usability effort is distributed into each activity resulting in the following effort variables:

- `Context of use effort.`
- `Specification requirements effort.`
- `Design effort.`
- `Evaluation effort.`
- `Revision effort`

4.3. Model Simulation

4.3.1. Scenarios definition

According to ISO standard 13407:1999 [11], before applying the UCD process it is necessary to plan it out, in order to specify how user centred activities fit in the overall development process.

To simulate the model three main scenarios have been considered. These scenarios will be mainly driven by the three usability levels considered. The `Usability level` and `Life cycle phase` input parameters will determine the values of `usability size` and `final usability effort` variables. At the same time, these variables will determine the initial distribution of usability tasks to be performed as well as the effort to be invested in them, setting the initial situation of the scenarios. The distribution of usability tasks in the life cycle, which reflects the assumed scenario for the simulation is detailed in figure 4.

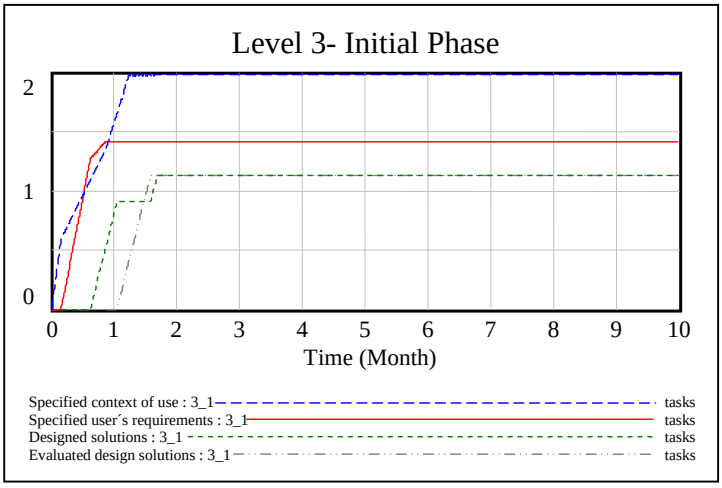


Figure 5. Results for the Initial phase and Level 3 of Usability

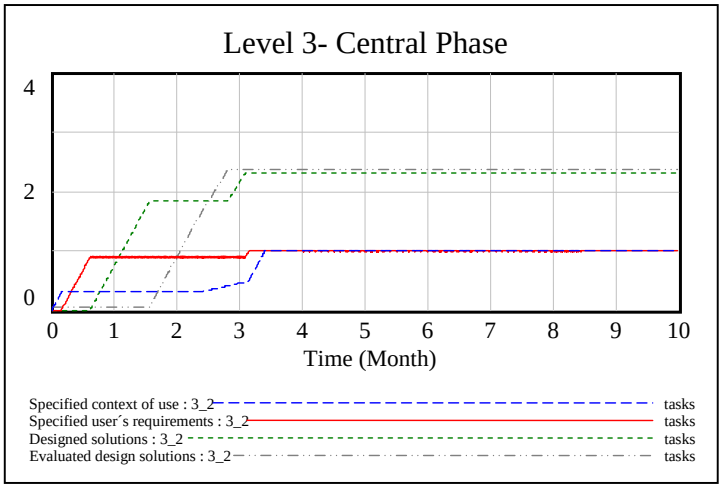


Figure 6. Results for the Central phase and Level 3 of Usability

Usability level 1 would correspond to the situation in which the usability methods and techniques are not correctly applied from the early stages of development, putting most of them off to the final stages. The initial scenario given by level 3 corresponds to an ideal situation in which usability activities would be taken into account through the whole life cycle of the web site project. Level 2 scenario defines an intermediate situation.

Once the `Usability Size` variable has been initialized, it is distributed -depending on the life cycle phase- into each of the variables corresponding to the number of UCD tasks that must be carried out in each process activity. Values are allocated according to those usability tasks that it is necessary to carry out in each activity during the application of the UCD process to the web design [2][3][8].

The `revision size` variable represents the percentage of evaluated tasks that need to be re-elaborated. This percentage increase as the level of usability decreases and the life cycle phase increases, since the number of errors encountered during evaluation is notably increased when usability is not taken into account since the early stages of development. [8][9].

Finally, the distribution of Usability Effort into each of the activities is carried out according to initial size of tasks for each activity, taking also into account revision tasks.

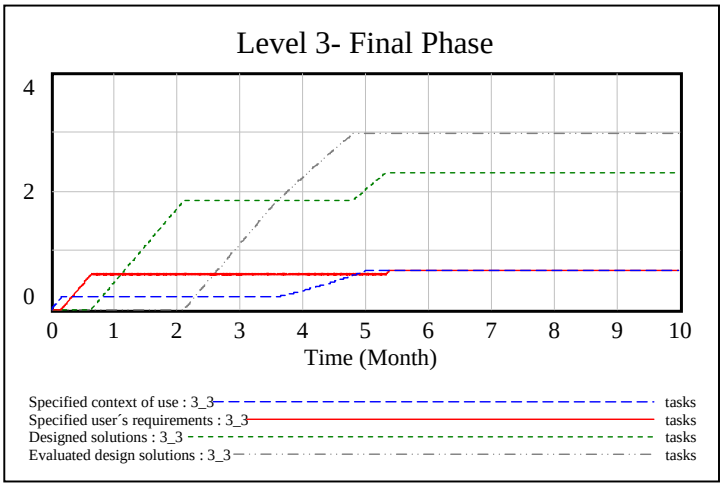


Figure 7. Results for the Final phase and Level 3 of Usability

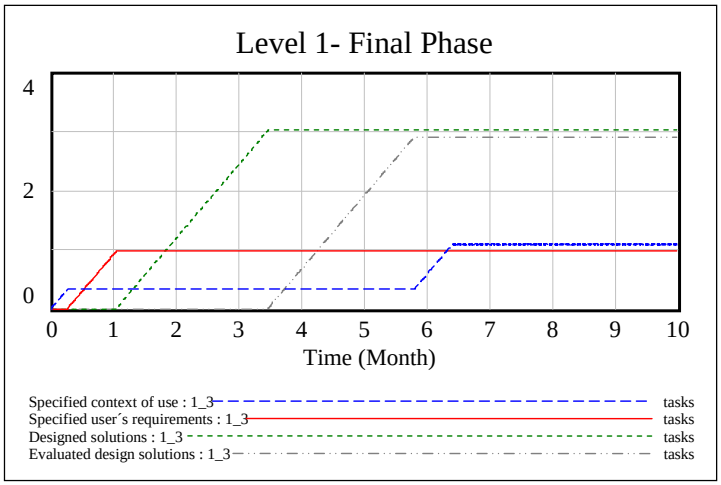


Figure 8. Results for the Final phase and Level 1 of Usability

4.3.2. Simulation Results

The model has been implemented using the Vensim® simulation environment. As an example, a simulation for a project of 11,000 SLOCs has been performed. Figures 5, 6 and 7 show the behavior of UCD activities for level 3 through the complete development process. The graphics show how the results reproduce the expected behavior from a qualitative point of view.

In figure 5 we verify that all UCD activities are involved in the initial phase. We can see that Use Context Specification activities as well as User´s Requirements Specification ones have a greater degree of importance in this initial phase of the life cycle, in which web site objectives are also planned and use scenarios defined. The curve corresponding to Solution Design represents the consideration of established guidelines for web writing style, navigation and page design as well as the design of early prototypes and mock-ups, which must eventually be evaluated in this phase for representative end users.

Figures 6 and 7 show how important design and evaluation activities become when ever more functional (and thus more complex) prototypes of the site are developed, their evaluation being consequently increased in complexity.

Figure 8 shows the behavior of the process for the final stage and for level 1 of usability. It is interesting to point out how development time is increased in comparison with level 3. This is chiefly due to the increase in the number of revision

5. Conclusions and Future Work

This paper presents the results of the application of simulation modeling to the UCD process. More precisely, the developed dynamic model helps visualize the behavior of UCD activities during the development life cycle of web site portal development.

Furthermore, it provides a tool to experiment the effects that the variations in the desired usability level and the estimated initial size have upon the UCD process evolution and behavior. Managers and developers could benefit from it to make decisions in order to improve the final product usability. The developed model is also useful to experiment with other types of software development projects.

The present paper contributes to justify the usefulness that modeling and simulation techniques – already validated in other software development paradigms- have to understand and improve the UCD process, setting a basis for its application in this scope.

Future research is oriented toward a deeper study of the application of modeling and simulation techniques to UCD integration into software development, as well as to the identification of the special features of UCD processes that help us model the specific aspects of usability methods and techniques, that affect interactive system usability both during the development process and in the evaluations of the final product once it is implanted.

Acknowledgements

The authors wish to acknowledge the Interdepartmental Commission for Science and Technology (Comisión Interministerial de Ciencia y Tecnología) Spain, for subsidizing this research through projects TIC 2003-369 and TIC 2001-1143-C03-02

6. References

[1] Abdel-Hamid, T., Madnick, S., *Software Project Dynamics: An Integrated Approach.* Prentice-Hall, Englewood Cliffs, NJ. 1991.

[2] Bevan, N., *UsabilityNet Methods for User Centred Design*. Human-Computer Interaction: Theory and Practice (volume 1 of the Proceedings of HCI international 2003). Lawrence Erlbaum Associates. 2003. pp. 434-438.

[3] Bevan, N., "Usability Issues in web site design". Proceedings of UPA'98.Washington DC. June, 1998. pp. 22-26.

[4] Boehm, Barry W., Chris Abts, A. Winsor Brown, et al. *Software Cost Estimation with COCOMO II*. Prentice Hall, 2000. pp. 51-55.

[5] Daly-Jones, O., Bevan, N., Thomas, C., "Handbook of user centred design". EC Telematics Applications Programme, Project IE 2016 INUSE, NPL Usability Services, National Physical Laboratory, Queens Road, Teddington, Middlesex, TW11 0LW, UK. January, 2001.

[6] Dix, A., Finlay, J., Abowd, G., Beale R.,. *Human-Computer Interaction.* Prentice Hall, Englewood Cliffs, NJ (2nd edition).1998.

[7] Ferré, X., "Integration of Usability Techniques into the Software Development Process". Proceedings of the Workshop Bridging the Gaps Between Software Engineering and Human-Computer Interaction. ICSE'03. (International Conference on Software Engineering). Portland, Oregon. May, 2003.

[8] Folmer, E., Bosch, J., Cost Effective Development of Usable Systems; Gaps between HCI and SE. Proceedings of the Workshop Bridging the Gaps Between Software Engineering and Human-Computer Interaction. ICSE'04. (International Conference on Software Engineering).2004.

[9] Harning, M.B, Vanderdonckt, J., Introduction to the proceedings of the workshop "Closing the gaps: software engineering and Human-Computer Interaction". INTERACT 2003.

[10] Hurtado N., Ruiz M., Torres J., "Aplicación del Modelado y Simulación de Sistemas Dinámicos al Proceso de Diseño Centrado en el usuario". Proceedings of the 5th ADIS 2004 Workshop on Decision Support in Software Engineering, University of Malaga. Published on CEUR-WS, vol 120. November, 2004. Available at: <http://CEUR-WS.org/Vol-120/>

[11] ISO 13407:1999. *Human-Centred Design Processes for Interactive Systems*. International Standard Organism, 1999.

[12] ISO 9241-11:1998. E*rgonomic Requirements for Office Work with Visual Display Terminals (VDTs)* - Part 11: Guidance on Usability. International Standard Organism, 1998.

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

[14] Madachy, R., "A Software Project Dynamics Model for Process Cost, Schedule and Risk Assessment". Ph.D. Dissertation. University of Southern California, Los Angeles, CA. 1994.

[15] Marcus, A. "Return on Investment for usable User-Interface Design: Examples and Statistics", Aaron Marcus & Associates, Inc, California, February, 2002, pp. 1-24.

[16] Mayhew, D., *The Usability Engineering Lifecycle*, Morgan Kaufmann, San Francisco, 1999.

[17] Nielsen, J.; Gilutz, S.. "Usability Return on Investment". Nielsen Norman Group, 2003.

[18] Reifer, D. "Web development: estimating quick-to-market software", IEEE Software, Vol. 17, No. 6.. 2000. pp.57–64

[19] Reifer, D. "Estimating Web Development Costs: There Are Differences". The Journal of Defense Software Engineering. June, 2002.

[20] Raffo, D., "Modeling Software Processes Quantitatively and Assessing the Impact of Potencial Process Changes on Process Performance". Ph.D. Dissertation. Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburg, PA. 1996.

[21] Raffo, D., Kellner, MI., Chapter 16. "Modeling Software Processes Quantitatively and Evaluating the Performance of Process Alternatives". En El Eman, K., Madhavji, N. (Eds.), Elements of Software Process Assessment and Improvement. IEEE Computer Society Press, Los Alamitos, CA. 1999.

[22] Ruiz M., "Modelado y Simulación para la Mejora de los Procesos Software". Ph.D. Dissertation. Department of Computer Languages and Systems. University of Sevilla, 2003. (In Spanish)

Economic Analysis of Integrated Software Development and Consulting Companies

C. Noujeim J. Sandrock and C. Weinhardt, *Department of Information Management and Systems, University of Karlsruhe*

***Abstract*— The still growing use of Internet technologies and corporate software shape the structure of the software industry. While visions of the Computational Grid or service orientated architectures promise systems and environments where users merely access software in a plug-and-play-manner, experience and empirical results point at the installation and integration efforts for software rollout and integration projects.**

Also economic practice blurs the boundaries between software and service providers and different models of software vendor and software integrator relationships have evolved. This paper therefore addresses strategic issues of joint software vendor/ software integrator companies with the aim to understand the underlying dynamics and complex feedback structures of such companies. With the help of a System Dynamics simulation model we analyze the complex interaction for different pricing, quality and timing strategies for the software production process in combination with several sales and software integration strategies.

***Index Terms*— Software development, service company, System Dynamics**

I. Introduction

In economic research software is typically considered to be an information product with distinct features such as lock-in effects, strong network economics, minimal marginal production (replication) costs and relative high creation and distribution costs [1]. However, the distinction between a product and a service is difficult to make, because the purchase of a product is accompanied by facilitating services including installation [2].

In economic practice, different models of software vendor and software integrator relations have evolved. On one hand independent software vendor (e.g. Siebel, Oracle) distribute their products and third-party software integrator (e.g. Accenture) complete the installation. On the other hand companies such as IBM, Siemens or SAP/SAP-SI offer single vendor turn-key solutions. Though the internet jeopardizes and des-intermediates traditional value chains[1] the latter organization model seems to resurrect in today's business life.

In this work, we consider an integrated software vendor/ software integrator company developing, marketing and distributing a new software product. We analyze the company's economic development and performance over a product-life-cycle of six and a half years.

Therefore we concentrate on the company's profit, measured in the cash flow over the relevant period. Revenue of the company is influenced by the product attractiveness offered by the company and hence determines profit and market share. We consider the software product attractiveness as a dynamic parameter affected by:

- **Quality attractiveness:** the quality attractiveness is divided into software product and software consultancy attractiveness and represents the degree of accuracy of the software code and the precision and excellence of the consulting service.
- **Pricing attractiveness:** the pricing attractiveness applies also to both software and consultancy pricing and foresees different pricing and licensing schemes including the four pricing policies with variable pricing, multi-dimensional pricing, the flat rate cap pricing, and flat rate pricing.
- **Timing attractiveness:** the timing attractiveness describes the effects of rapid time-to-market and early-mover advantages and includes as well the aspect of decreasing attractiveness for outdated software and the maintenance release time.

Therefore we analyze different pricing, quality and timing strategies. This should improve the management of such a company and stimulate the dynamic perception of causalities and effects of integrated software vendor/integrator companies. In the paper we will discuss in particular the effects of excessive sales, timing policies and segment differentiation strategies when the company concentrates it sales on a certain market.

II. Related Work

We consider the software development process as a project limited in time. And with System Dynamics, which is based on the principal of system thinking and is a tool that allows us to model business systems for today's complex market [4], we found an effective method to simulate and model software

[1] According to M.E. Porter [3] a value chain can be understood as a "collection of activities that are performed to design, produce, market, deliver, and support" products and services.

business.[2]

The domain of software simulation and modeling with System Dynamics offers a great variety of publications where several different models are presented. Constituting the fundamentals in the area of Software Project Management, T. Abdel-Hamid and S. Madnick developed the first software project simulation model with System Dynamics [5]. Their model consists of four elements: *human resource management*, *software production, controlling* and *planning*. Remarkable is also the model for software development of J. Duggan based on the Brooks' law [6]. It shows that adding extra manpower to a project can lead to an increased overall delay in the project delivery time [7]. Furthermore J. R. Madachy developed a dynamic model of an inspection-based software lifecycle process to support quantitative evaluation of the process [8, 9]. He built as well a business case model for a commercial software enterprise relating the dynamics between product development investments, software quality practices, market size, license retention, pricing and revenue generation [10]. In addition Cartwright and Shepperd introduce a model analyzing the dynamic behavior from a maintenance perspective [11].

These examples are the most considered ones in developing the model described in the following.

III. Simulation Model Overview

The focal company produces, markets, and distributes software products and consultancy services such as system integration and support. The model represents a single company facing stochastic, potentially unlimited demand and consists of the five following sectors.

1. Software Production
2. Software Integration and Consulting
3. Human Resources
4. Finance and Controlling
5. Software Market

A. Software Production

The Software Production sector examines software development and maintenance. These processes take a certain period of time constrained by the number of employees. The project size influences directly the project development time and the project complexity. Furthermore this sector considers different strategies of timing and quality influencing the software sales and company success.

- Timing Strategy models the effects that may accrue when the management decides to reduce the planed development time in order to accelerate the production processes and increases the time to market for the software product or a new release keeping the quality of the relevant product unaffected.
- Quality strategy: Here the management accepts the quality erosion of the product in order to reduce the development time.

Delays in the product delivery are not considered.

B. Software Consulting

The consulting sector describes the software support and services considering human resources and quality. It shows the impact of quality of software consulting on sales and vice versa. It considers also the estimated time for integration depending on the number of users in the enterprise and the product complexity. The consulting capacity is furthermore influenced by the number of enterprises (customers) which are supported. Also the quality depends on the number of customers and the productivity of the consultants. Therefore by means of interaction with Human Resources sector the module calculates the desired number of consultants.

It will show also the effects on sales, if the management decides to reduce the quality in order to reduce costs due to the consultants.

C. Human Resources

Human Resources sector is split in 3 parts:

1. Human Resources for software development: New employees will be hired, if the expected deadline can not be achieved with the actual number of employees. However while the resources are limited there are a maximum number of employees, which is pre-defined. The model calculates the appropriate number of the new employees to be hired. However the new employees must be trained in order to achieve the same productivity of experienced employees. This training process takes time and derogates slightly the production process in this period. The hiring and training time is modeled as a first order delay.
2. Human Resources for software maintenance: Very similar to the first part, new employees will be hired, if the deadline for software modification or new release can-not be met.
3. Human Resources for consulting: The company will hire new consultants, if the expected delivery time of the support ser-vice exceeds the customers' accepted de-livery time. Those new consultants, like the new employees of the software development, must be trained in order to achieve the same productivity as the experienced consultants. In addition avail-able human resources are limited. Due toe the training of new employees company's average support quality decrease with the number of new and inexperienced employees. If than the productivity of the consult-ants exceeds the demand, the

[2] According to J. Sterman [4] this approach is "a set of conceptual tools that enable us to understand the structure and dynamics of complex systems."

module will calculate and reduce the labor force until a minimum number of consultants, which is pre-defined. However the new consultants with lower productivity will be first decruited. This process is also modeled as a first-order delay.

Training time is assumed to be constant and voluntary quits or retirements are not considered.

D. Finance and Controlling

The finance sector tracks the performance of the software company. The cash flow is considered to be the most suitable measure of a company's success and is directly calculated as inflow minus out-flow. The inflow is the addition of the software product inflow and the consulting inflow:

- Software product inflow equals the product of the product price and the licenses sailed.
- Consulting inflow equals the product of the consulting price and the consulting days.

Customers pay immediately after delivery/consultancy without any price reduction or delay. The outflow covers wages and all other laboratory costs, rents, taxes, etc.

This sector presents also several pricing strategies for the software product including variable price model, multi-dimensional price model, flat rate cap price model and flat rate. The pricing of the product depends on the size of the customers (small, medium and large enterprises). We considered in our model that the variable parameter is the number of users in each customer company.

1. Variable price model: The price for the license is direct proportional to the number of the users. So it will give a constant average.
2. Multi-Dimensional price model: In this approach, companies pay additionally to the basic price – which is a fixed price – a variable license fee which is directly proportional to the number of users (employees of the company). Hence, the average price per user will constantly decline with the increasing number of users.
3. Flat Rate Cap model: The price will be constant until a defined number of users and than a variable element will be added. The outcome of this is also that the aver-age price declines with the increasing number of users.
4. Flat Rate model: In this model, the price is set as a constant and does not depend on the number of users. The average price will decline rapidly with the increasing number of users.

Additionally it presents the same pricing strategies models for the consultancy, however the integration efforts (days) are the variable parameter.

E. Software Market

The relevant market is split up into three parts – small, middle and large enterprises – in order to achieve an adequate granularity for each part. The module considers the effects of sales on market share which again impacts sales.

In order to simulate scenarios where the company already exists on the market, we set for each market sector a number of concurred customers using an older version of the software product. Thereby we determine market share of the local company prior of the new product launch and hence strongly influence the "System Lock In". Sales for the new product on the other hand depend on its attractiveness and the "System Lock In" of the company. Hax and Wilde characterize three Business strategic options [12]:

- Best product – competition based upon product economics: low cost or differentiation.
- Total customers satisfaction – competition based upon customer economic: Reducing customer costs or increasing their profits
- System Lock In – competition based upon system economics: complementor Lock In, competitor Lock out, proprietary standard.

The "System Lock In" effects are crucial in software providers' business strategy process. For instance Microsoft Office has a strong "System Lock In" because of the immense customer base and the ubiquity of office application it has become a de facto standard within most companies. Hence most documents must be authored or converted into Office readable documents. Furthermore, "Switching costs" are normally high in the Software Business and are considered in our model: if customers acquiring the new product will not switch to a competing vendor within the next 4 ½ years.

F. The Company Model

The model, which consists of 339 equations, elucidates the dynamic relations and feedbacks between all its components. In this connection the model simulates the impact of different pricing strategies on sales. It provides also many scenarios, e.g. the impact of timing on software development cost and sales as well as sales on software support, quality and cost. Besides it shows the impact of excessive product sales on software support quality which influences the software product sales. In this paper will discuss 3 scenarios:

1. Scenario 1: the management will set the price for software product and for software consulting low, avoiding here a "price war" scenario.
2. Scenario 2: the management will decide to bring the product 6 months earlier to the Market.
3. Scenario 3: the management will decide to concentrate on the middle and big enter-prices.

IV. VALIDATION

The structure of the model is inferred from intensive literature research including the above mentioned literature and software development guidelines and standards such as V-Model. The Human resource module for instance is based on

TABLE I
SENSITIVITY ANALYSIS

	Value	Cost		Cash Flow		Revenue Consulting		Revenue Software	
		Deviation	RSD*	Deviation	RSD	Deviation	RSD	Deviation	RSD
Accepted Delay	2,5 month	5,48%	0,90%	-25,90%	4,98%	-2,68%	0,33%	-5,79%	0,33%
Salary Consultants	3000; 4000 EUR / empl.	6,09%	2,71%	-27,96%	10,50%	-2,80%	0,00%	-5,91%	0,00%
Price Consultancy[a]	:	-0,48%	4,95%	-8,52%	25,15%	-1,81%	8,28%	-3,89%	8,36%
Quality[b]	1	-5,57%	3,02%	-28,94%	25,45%	-11,94%	7,91%	-12,08%	8,20%
Initial Employees Production	40 empl.	-0,90%	0,83%	-10,03%	14,25%	-2,97%	3,72%	-3,18%	3,88%
Training Time Development	3 month	5,84%	0,05%	-27,30%	0,19%	-2,80%	0,00%	-5,91%	0,00%
Salary Maintenance	3000; 4000 EUR / empl	5,79%	0,98%	-27,17%	3,71%	-2,80%	0,00%	-5,91%	0,00%
Initial Employees Consultancy	40 empl.	0,00%	0,20%	0,01%	0,22%	0,00%	0,08%	0,00%	0,10%
Price Product[c]	:	1,76%	4,71%	-9,49%	36,66%	-0,86%	13,59%	-2,27%	9,16%
Employees Maintenance	10 empl.	-1,27%	2,62%	-17,74%	35,32%	-5,86%	7,80%	-5,88%	7,98%
Salary Development	3000; 4000 EUR / empl.	5,74%	1,89%	-27,05%	7,32%	-2,80%	0,00%	-5,91%	0,00%
Target Time to market	24 month	-7,75%	2,56%	-77,08%	26,28%	-28,36%	0,56%	-25,26%	1,47%

* Relative Standard Deviation
[a] Prices vary according to the target customer segment: for small sized [4000, 7000], medium sized [7000, 10000] and large companies [10000, 13000]
[b] Quality varies between [0.8, 1] since the quality is bounded by 1.
[c] Prices vary according to the target customer segment: for small sized [1000, 4000], medium sized [4000, 9000] and large companies [90000, 14000]

Brooks' law [7]. Furthermore structures of familiar models or (sub-) models are considered and incorporate or modified when applicable. Some parameters such as salary per employee can be assessed by available empirical data or market and industry reports.

Unfortunately, the structure of the model has not been validated empirically with interviews; calibration of the model is constrained by the model's size and by the scarcity of empirical evidences. Behavior reproduction tests or Turing tests have not been carried out. The presented model therefore must be considered as a preliminary model a starting point for further empirical validation [13].

However, extreme behavior tests (e.g. production, time, hiring times etc.) and intensive numerical sensitivity analysis support the robustness of the model. The influence of most of the initial parameters has been assessed within the symmetric interval of -10% and +10% of the initial value in a univariate sensitivity analysis; salaries for consultants, developers and for the maintenance force have been studied with a multivariate analysis. The impact of the parameter variation accounts in general for at most 6% of the cost development, 3% of the license revenue and 6% of the consultancy revenue deviation form the reference case after six years. However, the targeted time to market of the software product has a stronger influence on both cost and revenue – This case will be discussed in Scenario 2.

Some results of the sensitivity analysis are depicted in Table 1.

V. SIMUALTION SCENARIOS

A. THE BASIC SCENARIO

In this scenario the management of the company decides to develop a new software product and to bring it on the market within two years. In the beginning the project size is fixed at 8000 (working/day). The company starts with 40 trained employees and each of them has a productivity of 8 (hours/day). The model calculates the need of new resources and hires them (figure 1). However new employees have a limited productivity of 2.4 (hours/day) and need to be trained in order to achieve the full productivity. The training process takes three months. During this training process the productivity of the trained employees also decreases because they are training the new employees. Figure 2 depicts the progress of the product development process. The product marketing starts when 80% of the product development is finished. The products maintenance starts immediately after the product is been launched and the releases and updates are scheduled in this scenario every twelve months. The company has 50% of the market share. We model the concurrent price of the software product and consultancy on the market as a

random geometric walk. The management decides to use for software pricing a flat rate pricing model for the three market sectors. The expected price equals the mean value of the upper and lower bound of the chosen truncated normal distribution. The management uses a variable pricing model for consultancy services and similarly the average price, which is the product of the price and the number consultancy days, takes also in the mean value of the extremes of the random function. This makes the average attractiveness of the product price and the consultancy price around 0.5 (with 1 as maximum). Due to the sales the company needs consultants to integrate and support the product. The initial number of trained consultants is set to 40 employees. Because of the limited available resources on the market, we define the maximal number of consultancy employees on 80. If new consultants are needed, the module will calculate and hire them. The training period is also three months. Figure 3 shows how many consultants are needed in order to keep the quality of consultancy 1 (with 1 as the highest quality) while figure 4 depicts the number of trained consultants (productivity = 8 hours/day). Needing to train the new employees (2.4 hours/day), which causes also production erosion, a quality fall cannot be avoided (figure 5). The cash flow of this scenario is shown in (figure 6) and the market share for the three market sectors is presented in figure 7.

In figure 1 we can see that the model hires at the beginning two new employees in order to bring the product on time (time=24) as shown in figure 2. When the product is launched the consulting begins. At the launch time of the product the attractiveness of the product starts very high with 1. Then it decreases as a *Smooth Function* with the time until a new release is launched which affects the attractiveness in a positive way. We see in figure 3 that the need for consultants is much higher than 40 and exceeds also the maximum number of resources. This is why 40 new consultants are hired and the number of trained employees rises with time as shown in figure 4. The quality of consultancy suffers from the lower production (figure 5). The quality is modelled as a Smooth Function, because bad quality needs time to recover. This affects the cash flow and the Market Share.

- Cash Flow at (time=78) is 7.32842 Million Euro
- Market Share (time= 78) is 48.9953 % (this is due to the lower attractiveness of the product at the end of its life cycle).
- Market Share Small enterprises = 48.7651%
- Market Share Middle enterprises = 51.4553%
- Market Share Large enterprises = 48.1969%

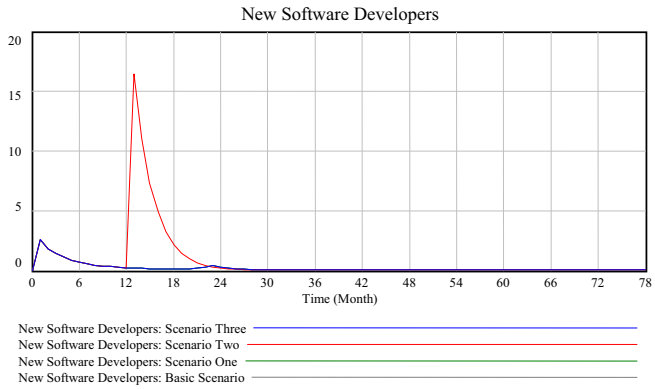


Fig. 1: New software employees

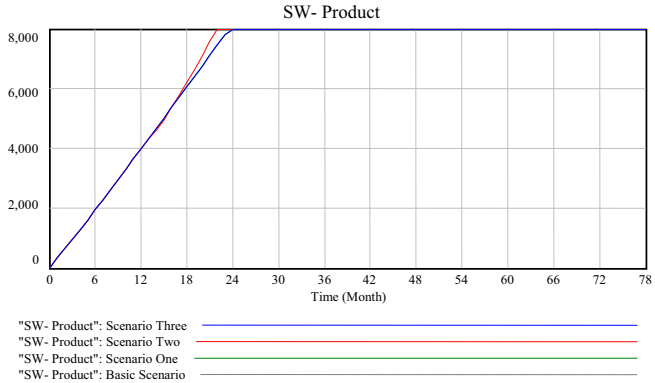


Fig. 2: Software product development

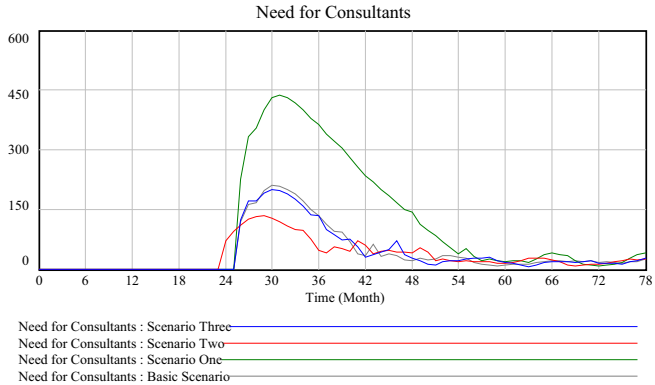


Fig. 3: Need for consultants

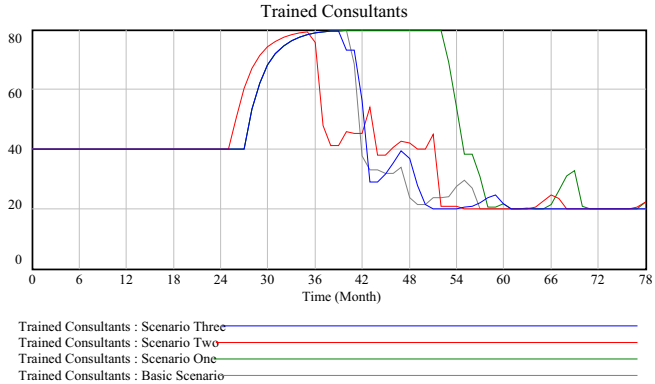


Fig. 4: Experienced consultants

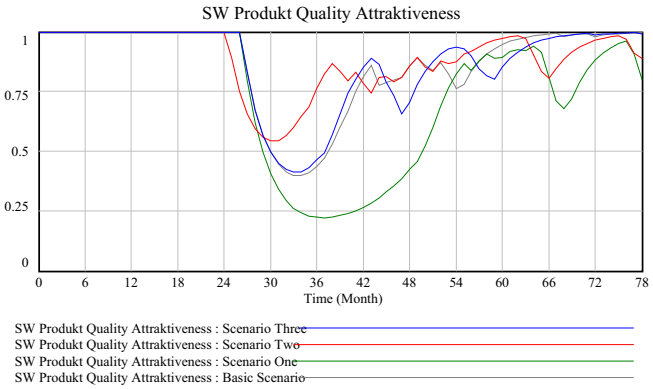


Fig. 5: quality attractiveness

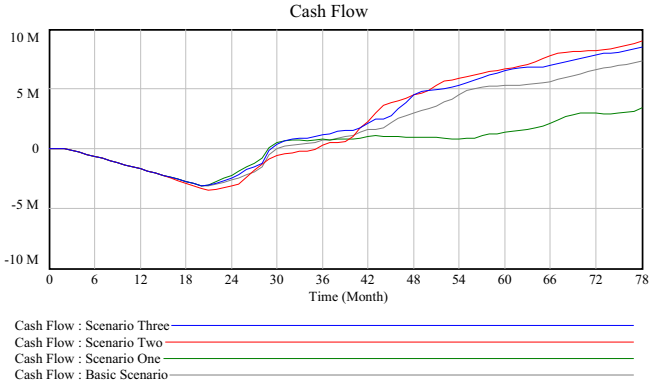


Fig. 6: Cash Flow

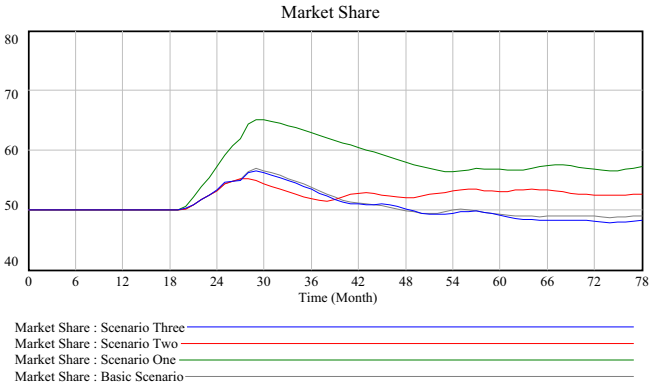


Fig. 7: Market Share

SCENARIO 1:

In Scenario 1 the management decides to use an aggressive pricing strategy in order to capture a higher market share (80%). They set the prices for software by using a variable price model for small enterprises, a multi-dimensional price model for middle enterprise and a flat rate cap model for the large enterprises. The prices for consultancy are defined for the three markets segment with variable pricing models. The prices however are around the lower extreme of the concurrent price random function. Here we don't consider any "game scenario". While the prices are low, the pricing attractiveness of the product for both software and consultancy is around 1 (with 1 as maximum). This implies that the sold licences are much higher than in Basic Scenario (figure 8). In addition this implies that the need of consultants is much higher (figure 3). This affects the quality of consultancy (figure 5) which affects on its part the sales. This is why the market share, which increased at the beginning of the sales, decreases until the quality recovers (figure 7). However, when the quality recovers the attractiveness of the product due to time was lower this is why market share does not increase again. This scenario shows that the strategy chosen did not achieve its goal.

- Cash flow is 3.38 Million Euro
- The Market Share increased only to 56.9805%
- Market Share Small enterprises = 56.2093%
- Market Share Middle enterprises = 58.9246%
- Market Share Large enterprises = 56.2093%

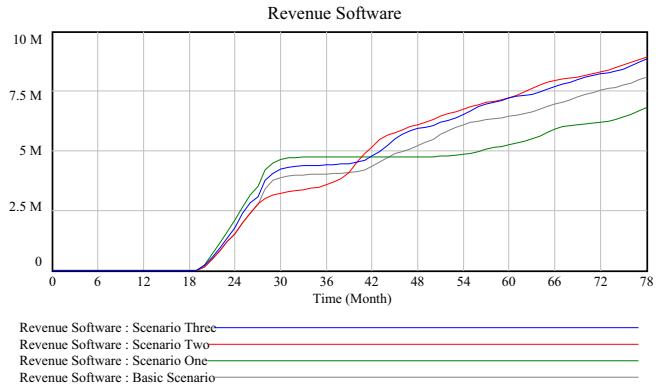


Fig. 8: Software revenue

SCENARIO 2:

In this scenario the pricing strategy is the same as in the Basic Scenario, which implies an average price attractiveness of 0.5. Furthermore the management decides twelve months after the beginning of the project to launch the product six months earlier in order to get a higher market share due to a very innovative product on the market. The model calculates and hires the new developer as shown in figure 1. This curve reach a maximum of 16 new employees at time = 13. However due to training time and production erosion the product is finished at time = 22 (figure 2). This new product, which is released two months earlier than in the Basic Scenario, has a very high timing attractiveness and a time advantage. The time advantage of two months has four positive feedbacks:

- The consulting of the product can begin two months earlier. This influences the Cash Flow.
- The attractiveness of the product is higher due to the product innovation.
- The product life cycle is also two months longer.
- The marketing of the product starts also in this scenario at time=18 (when 80% of the product is finished) (figure 9).

However, the investment for this product due to the higher number of software developer is higher as shown in figure 9. As the period between the marketing of the product, which start at time 18, and its launch is smaller (two months), the sales which happen in this period are less than the sales which happen during the same period in the Basic Scenario. On the other hand the consulting of the product begins earlier. This means that at the beginning of the product the need of the consultants is smaller as shown in figure 3 which also affects the quality of the consultancy as shown in figure 5. Higher timing attractiveness, consultancy quality attractiveness and a longer product life cycle affect licenses sale and cash flow and also Market Share. However approaching the issue of reducing the time of development needs to be well studied and is not very simple due to the project coordination difficulties. Other work mentioned in the literature approaches this area. Project development needs time and many resources and since the company has amongst others limited resources, the management can not reduce the development time as it wished to. In this Scenario we show that the initial strategy of the management to launch the product six months earlier failed. This is due to Brooks' law which says that adding additional manpower to a software project often make it later, because of the project coordination difficulties [7].

- Cash Flow = 9.02003 Million Euro
- Market Share = 52.607 %
- Market Share Small enterprises = 51.4894%
- Market Share Middle enterprises = 55.0758%
- Market Share Large enterprises = 51.8039%

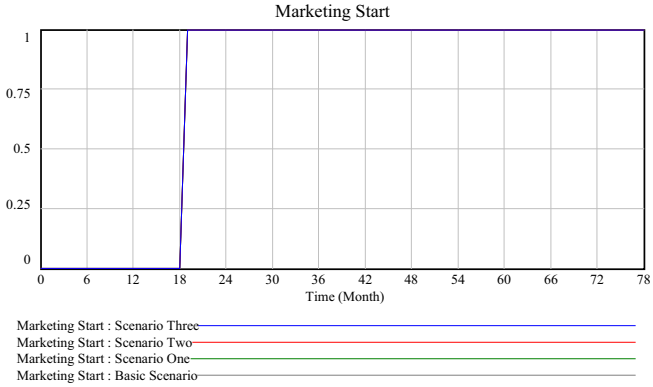


Figure 9: Marketing start

SCENARIO 3:

In this scenario the management decides to concentrate on middle and big enterprises. They set the price as in Table II:

As we see in table II the attractiveness of the price for small enterprises is low and that is why the market share of the small enterprises drops to 41.466% (figure 10). However, due to the high switching cost and the System Lock In staying

TABLE II
PRICING FOR SCENARIO 3

	Small enterprises	Middle enterprises	Large enterprises
Software Price	Multi-Dimensional Fixed Price = 1000 Euro Variable Price=100 Euro	Flat Rate Price = 5500 Euro	Flat Rate Price = 10500 Euro
Consultancy Price	Variable Price Price = 1500 Euro/ Day	Flat Rat Price = 8500 Euro	Flat Rat Price = 12000 Euro
Average Software Price Attractiveness	0.15	0.73	0.72
Average Consultancy Price Attractiveness	0.17	0.5	0.33

.

almost constant, 60% of the initial small enterprise customers decide to pay the high price which lies in average on the higher extreme of the concurrent price Random Function. The market share for the middle enterprises rises to 54.7323% because of the attractiveness of the software price which lies below the average price of the concurrent price (figure 11). The market share for the large enterprises is 49.9408% (figure 12). This is by reason of the average price of software and consultancy which is 0.525 and the lower time attractiveness at the end of the product life cycle. The quality of consultancy is better as in the Basic Scenario (Figure 5). This, the higher software and consultancy price for small enterprises and the higher number of licenses sold for the middle enterprises, force up the cash flow to 8.5 Million. This Scenario shows that due to the limited resources available for the software company, especially the consultancy resources, the company can not optimise the consultancy quality with such resources. Concentrating the resources to a certain segment of the market can raise the Cash Flow.

- Cash Flow = 8.50561 Million Euro
- Market Share = 48.1635
- Market Share Small enterprises = 41.466%
- Market Share Middle enterprises = 54.7323%
- Market Share Large enterprises = 48.9408%

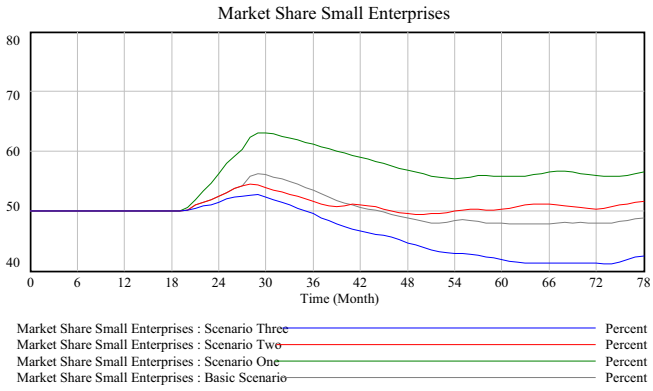


Fig. 10: Market Share for small enterprises

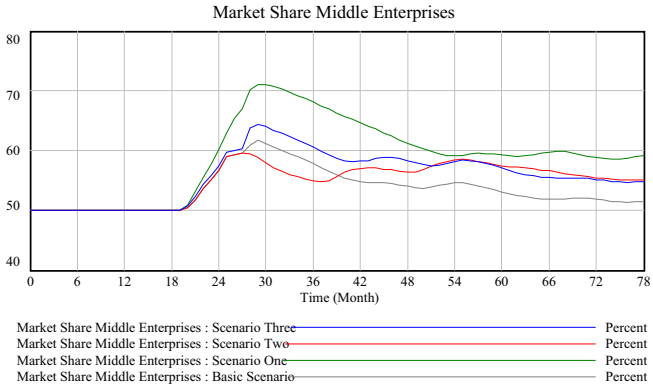


Fig. 11: Market Share for Middle Enterprises

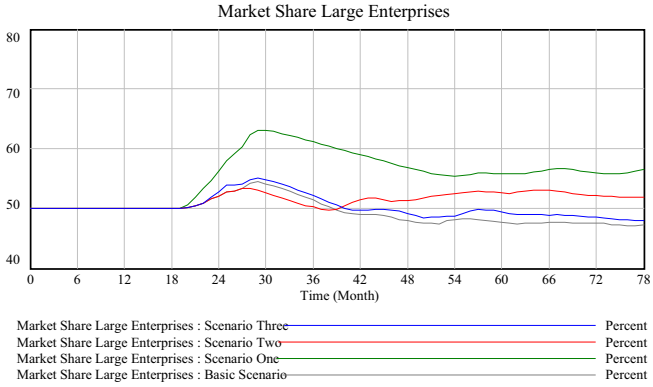
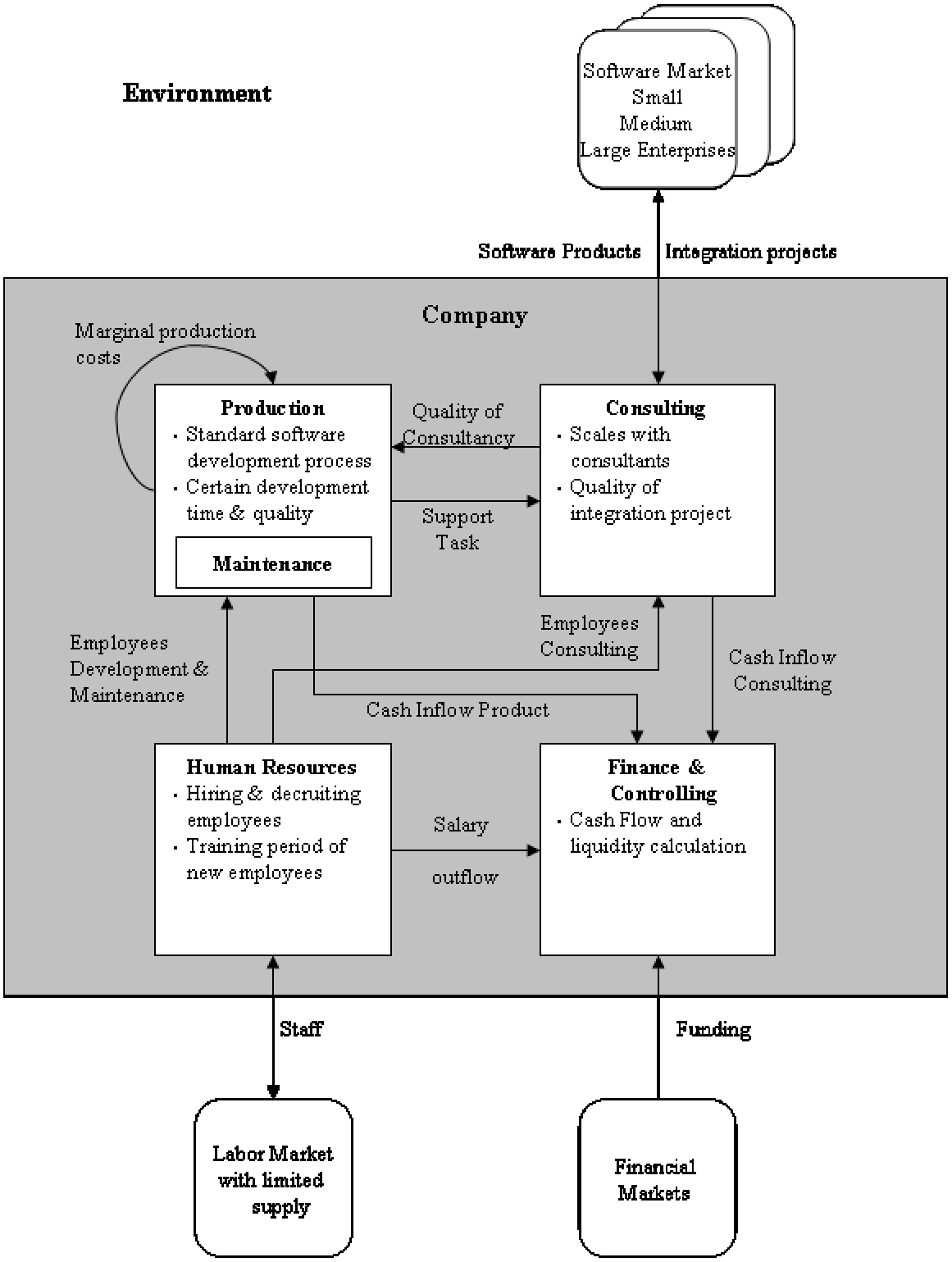


Fig. 12: Market Share for large enterprises

VI. Conclusion

Those scenarios illustrate the independences between the quality of software consultancy and the cash flow of the product. Limited resources for the consultancy support of a software product will decrease enormously the cash flow of a company, if excessive sales happen. Furthermore the market share can not achieve its growth target. We applied also Brooks' law on the software development and showed its impacts on the company market share and cash flow. Bringing a project earlier on the market costs more investment but increases the revenue. And finally we discussed the case when the management decides to concentrate on a Market segment, using different pricing strategies. Due to the limited resources of the company, this procedure implies a higher market share in the chosen sectors and a higher Cash Flow. However, we didn't discuss exclusively in this paper the cases of using different pricing strategy, although this model is able to analyse this.

The simulations for different pricing and licensing scenarios reveal the complex feedback structures in the price setting process and its impacts on the performance of the integrated software and consulting company. Further research efforts should be made towards validation and calibration of (sub) models though the size of the model limits the value of a carefully calibrated model. Empirical data for instance from company or industry reports have not been incorporated in the modelling process but would be a desirable enhancement. Behaviour reproduction or Turing tests have not been carried out but should increase the confidence in the model.


Environment
Software Market
Small
Medium
Large Enterprises
Software Products
Integration projects
Company
Marginal production costs
Production
· Standard software development process
· Certain development time & quality
Maintenance
Quality of Consultancy
Support Task
Consulting
· Scales with consultants
· Quality of integration project
Employees Development & Maintenance
Employees Consulting
Cash Inflow Consulting
Cash Inflow Product
Human Resources
· Hiring & decruiting employees
· Training period of new employees
Salary
outflow
Finance & Controlling
· Cash Flow and liquidity calculation
Staff
Funding
Labor Market with limited supply
Financial Markets

REFERENCES

[1] Shapiro, C.; Varian, H. R.: Information Rules: A Strategic Guide to the Network Economy, Boston: Harvard Business School Press, 1998

[2] Fitzsimmons, J. A.; Fitzsimmons, M. J.: Service management, 4. ed., Boston: McGraw-Hill, 2004

[3] Porter, M. E.: Competitive Advantage, New York: Free Press, 1985

[4] Sterman, J. D.: Business Dynamics, Boston: Irwin McGraw-Hill, 2000

[5] Abdel-Hamid, T.; Madnick, S.: Software projects dynamics - an integrated approach, Prentice-Hall, 1991

[6] Duggan J. : System dynamics simulation for software development, National University of Ireland, Galway 2004

[7] Brooks F.P.: The mythical man-month, Addison-Wesley Publ., 1995

[8] Madachy R.: A software project dynamics model for process, cost, schedule, and risk assessment, PhD Thesis, University of Southern California, Los Angeles, 1994

[9] Madachy R.: System dynamics modeling of an inspection-based process, IEEE Computer Society Press, Berlin, 1996

[10] Madachy R.: A software product business case model, USC Center for Software Engineering and Cost Xpert Group, Prosim 2004

[11] Cartwright M; Shepperd M.: On building dynamic models of maintenance behaviour, in Proc. 10th European Software Control and metrics Conf. Shaker Publishing, Hertmonceux, 1999.

[12] Hax A. C., Wilde D.L.: The Delta Project: Discovering new sources of profitability in a networked economy, St. Martin's Press, 2001

[13] Vennix, J. A.: Group model building, Chichester: Wiley, 1996

Part 3

Experience Reports

Implementing Generalized Process Simulation Models

David M. Raffo+*, Umanatha Nayak*, and Wayne Wakeland[&]

+College of Engineering and Computer Science
*School of Business Administration
[&]Systems Science Program

Portland State University
Portland, Oregon, USA

Abstract

Over the past decade, software process simulation has been used by both researchers and practitioners to assess the cost, quality and schedule associated with projects. Although this technology has been shown to have success in addressing questions ranging from strategic management issues to process improvement planning as well as management training, this technology has not yet been widely adopted. One of the inhibiting factors is the cost of developing these models. In order to reduce the cost associated with developing software process simulation models, we have developed generic building blocks that can be used to quickly create software process models. In addition, our software process simulation blocks can offer important benefits in terms of improving the quality and usability of these models. In this paper, we present our generalized process simulation model (GPSM) concept and blocks. We also present two illustrative examples that apply the generalized blocks to create software development lifecycle models at two leading organizations.

Key Words: Software Process Modeling, Process Reuse, Process Simulation, and Generic Blocks.

1 Introduction

Software process simulation (SPS) is an emerging technology/methodology that can be used in software project management to estimate project-level performance, carry out 'what-if analyses', and predict the impact of process changes among others. Raffo et al. [1] state that industry has yet to take full advantage of the benefits offered by SPS models (SPSMs). They cite two main reasons for this:

- Process models tend to be difficult to build and maintain.
- The metrics data for the process models can be difficult to acquire (or may not exist).

The general perception is that SPSMs are costly to build and that timely results are difficult to achieve. SPSMs may be developed at different levels of scope and depth to suit an organization's needs. The higher the level of detail and fidelity to the process, the greater amount of effort required to build the SPSM. Further, the simulation tools used to develop an SPSM can have a significant impact on the amount of effort and the degree of expertise required. Most simulation tools are very general and are not tailored to address software process modeling requirements. These requirements include such concepts as modularization, reuse, and product families. The following research is of particular interest:

- information hiding by Ribo et al. [9],
- the modularization concept by Parnas [2 – 6], and
- the product family concept by Weiss et al. [7].

Given the obstacles to the adoption of SPS models by industry discussed above, and the results from prior research, Raffo et al. [1] believe a tool is needed that provides an architecture and model structure in the form of templates and basic model building blocks. Such a tool would significantly reduce the expertise required of the modeler, model analysts, and model users. They propose a generalized process simulation model (GPSM) component logic with the following modularization framework provided by Parnas [2 - 6]:

- Partitioning the Simulation Process Logic
  - Reducing Logic (Complexity)
  - Protecting Logic (Errors)
- Changing the Simulation Process Logic
  - Internal (High Changes)
  - Interface (Low Changes)
  - Grouping (Manageable Aggregation)
- Understanding the Simulation Process Logic

- Document Internal
- Document Interface
- Relevant Access

Raffo et al also adopt the "product family" concept as defined by Weiss et al [7] within the GPSM framework. The product family defines the overall GPSM domains by the processes being modeled, by the facilities that will be required to promote the development of family members, and by the ability to rapidly create these members into usable simulation applications that perform within their context.

Starting with this broad concept of GPSM, this paper proposes a specific framework for building software process simulation models by making use of generic building blocks. First, we discuss the different activities involved in software development, and how these activities can be partitioned into four software development activity categories. Then, we present example generic model blocks designed to support the creation of SPSMs. Next we discuss example high-level process/activity blocks built using the GPSM generic blocks. Finally, we briefly describe two models built using a combination of GPSM process/activity blocks and generic model blocks.

2 Building GPSM Product Platform

This section addresses some of the implementation-related issues of generalized process simulation model (GPSM) concepts. This work makes use of Extend™ from ImagineThat, Inc. to implement these concepts and addresses following concepts involved in creating GPSM product platform:

- Categorizing software development activities,
- Building generic GPSM blocks,
- Building GPSM process/activity blocks.

Each of these will be discussed further in the following sub-sections.

2.1 SW Development Activity Categories

We can classify various SW development activities into following four categories:

- **Development** – where an entity is being worked on or developed and a likelihood of some defects being injected into it. This block can be used to represent requirements, design, coding, or other forms of development activity.
- **Inspection** – where an entity is being reviewed or inspected to detect the defects injected during the development activity.
- **Rework** – where an entity is being reworked to correct the defects detected during the inspection activity and a likelihood of some bad-fix type of defects being injected into it.
- **Testing** – where an entity is being tested to detect any defects that may have been injected during development and rework activities.

Classification of various SW development activities into four distinct categories helps with modularization, reduction of complexity, protection of logic, and reuse of GPSM components. We can further classify the Testing activity into four sub categories namely: Identify Anomalies, Verify Anomalies, Rework, and Regression Testing.

- **Identify Anomalies** – where an entity is being tested on to detect the defects. This testing may detect one or more anomalies that point to one ore more underlying defects during the verification process.
- **Verify Anomalies** – where an anomaly detected during Identify Anomalies activity is traced to the underlying software defect (one or more anomalies might be caused by a single underlying defect).
- **Regression Testing** – where a reworked entity is being tested using a subset of test cases to make sure the entity performs correctly as expected.

The Inspection activity can be further divided into two or more low level activities such as Inspection Preparation, and Hold Inspection Meeting.

- **Inspection Preparation** – where an entity is being prepared for an inspection activity.
- **Inspection Meeting** – where a meeting takes place to evaluate or inspect an entity.

Defining these low-level inspection activities enables us in creating GPSM process

blocks for Inspection types like Walkthrough, Desk Checking, and Full Fagan etc.

2.2 Building Generic GPSM Blocks

The generic GPSM blocks greatly reduce the complexity involved in building software process models. In order to manage these generic blocks better and facilitate component reuse, a new Extend™ library called SW, Generic was created to hold these generic GPSM blocks. Some of the new blocks created in this library are:

- Activity, Development,
- Activity, Inspection,
- Activity, Rework,
- Activity, Identify Anomalies,
- Activity, Verify Anomalies,
- Activity, General,
- SW, Resource Pool.

The generic GPSM blocks work in conjunction with SW, Resource Pool block, a new block that handles the allocation, release, and preemption of resources to various activities.

2.3 Building GPSM Process/Activity Blocks

Once the generic GPSM blocks were designed and developed, the following GPSM process blocks were created by utilizing generic GPSM blocks from SW, Generic library making use of hierarchical block concept of Extend™:

- SW, Testing,
- SW, Joint Review,
- SW, Inspection (Formal).

These new GPSM process/activity blocks were placed in new GPSM library called SW, Process. As discussed in the previous section, creation of these blocks was necessitated by the fact that some of the activities had multiple sub tasks in them. In addition to this, we can add other GPSM process/activity blocks to this library as the need arises. *Figure 1* shows a screen shot of a SW, Inspection (Formal) block while the SW, Testing block is shown in *Figure 2*.

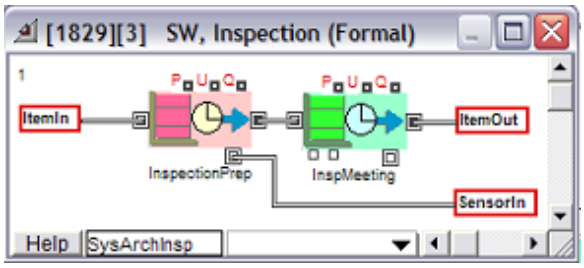


Figure 1: Screen shot of SW, Inspection (Formal) block

3 Building GPSM Models

Using these new GPSM libraries (i.e. SW, Generic and SW, Process), it is much easier to develop life-cycle models for real software development projects. To demonstrate the ease of use and reduction in cost, we built the following lifecycle models using the new GPSM libraries:

- A tailored IEEE12207 software process with an IV&V process being used at NASA,
- An incremental development process used at a leading software development firm.

In the first model (shown in *Figure 3*), we added an IV&V process layer along with a tailored IEEE12207 process model for large-scale NASA projects. This model contains 86 process steps and has 3 levels of hierarchy to support both systems and software development projects. Using the generic blocks provided in the GPSM environment, process steps might be easily added or removed so that the model can be easily tailored to specific company processes.

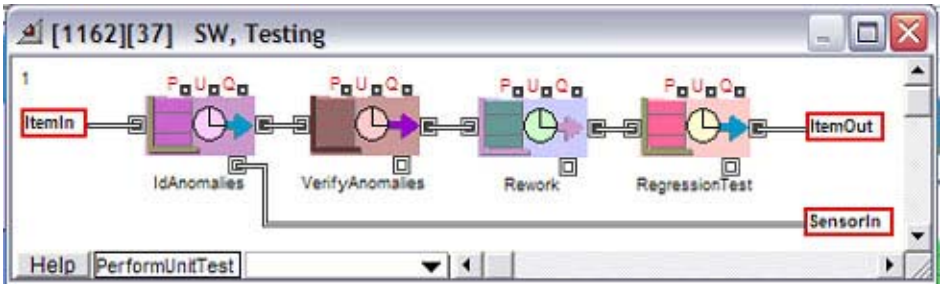


Figure 2: Screen shot of SW, Testing block

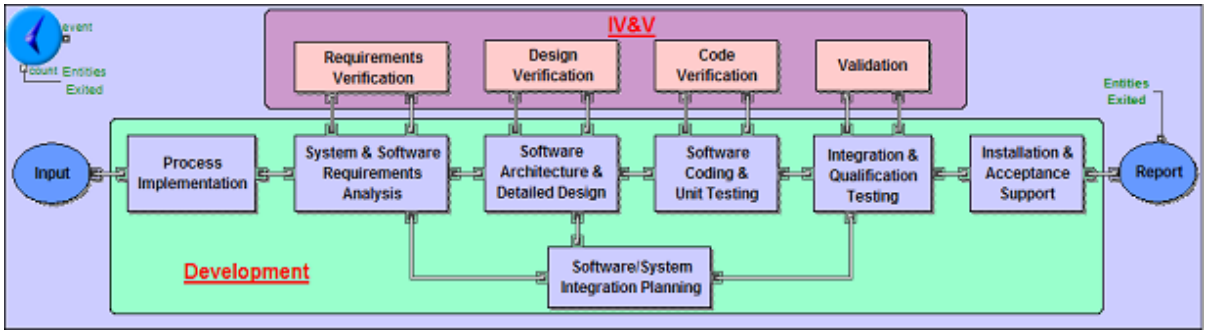


Figure 3: Screen shot of a tailored IEEE12207 software process lifecycle model

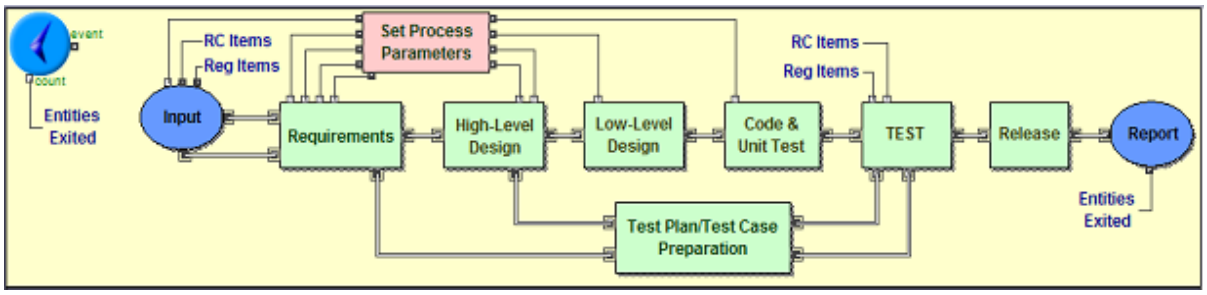


Figure 4: Screen shot of an incremental development process model at a leading SW firm

In addition, *Figure 4* shows the model of an incremental development process used at a leading software development firm. This model also has multiple layers of hierarchy all built from the generic process blocks described above. These two models confirmed that the custom libraries significantly reduced the time to build the models and hence reduced the cost. Both the IEEE12207 software lifecycle process and the incremental development lifecycle models were built in two days rather than several weeks that would have been required to build such a model without the new GPSM libraries.

Many favorable comments were provided by users regarding the usefulness and appropriateness of GPSM. Some of them are:

- Very easy to construct the model using GPSM blocks compared to using regular Extend™ blocks.
- GPSM blocks hide the complexity from the users.
- Took a couple of hours to understand how GPSM blocks work (of course previous experience with Extend™ assumed).
- Once we had the data, we required only a couple of hours to populate the model.
- Requires less memory and runs faster.

4 Conclusions

This work demonstrates that specific SPSMs can be built very quickly when using the GPSM framework – a set of generic software specific building blocks. These generic GPSM blocks enable modularization, reuse of components, and creation of product families. The demonstration included constructing models for a tailored IEEE12207 software life-cycle process and an incremental software development life cycle. These models were built in hours rather weeks and GPSM framework was very favorably received by users who were well versed in alternative methods.

The use of GPSM concepts and GPSM building blocks makes it very easy to build SW lifecycle models and leads to a substantial reduction in time and cost of building SPSMs. Once we build these models, we can run various scenarios or use cases by changing various model parameters. In addition, these models can be used as templates and quickly adapted to other projects within the firm. In future work, we plan to add more blocks to the libraries and build models for other software life cycles using this GPSM framework.

Acknowledgements

This work has benefited from programming work done by Bhuricha Sethanandha and testing work carried out by Siri-on Setamanit whose contribution is gratefully acknowledged. In addition, we are grateful to NASA for supporting this research effort under grant NAG5-12739.

Biographies

David Raffo

Dr. Raffo is currently Associate Professor of Computer Science and Business Administration at Portland State University. He is also a Visiting Scientist at the Software Engineering Institute. Raffo completed his Ph.D. at Carnegie Mellon University. Dr. Raffo's research interests include: Software Process Design, Financial Analysis of Software Engineering Decisions, Process Simulation, and Value Based Software Engineering. Dr. Raffo has over forty refereed publications in the field of software engineering and is co-Editor-in-Chief of the international journal of *Software Process: Improvement and Practice*. He has received research grants from the National Science Foundation (NSF), the Software Engineering Research Center (SERC), NASA, IBM, Tektronix, Motorola and Northrop-Grumman. Prior professional experience includes programming as well as managing software development and consulting projects at Arthur D. Little, Inc. Dr. Raffo teaches a variety of courses in Software Process and Operations Management.

Umanatha Nayak

Mr. Nayak has over eleven years of software development experience working as a SW engineer, senior systems Analyst and IT Consultant. He has worked for companies like British Telecom, Pacific Telecom, CenturyTel, Alaska Communications Systems, and Mahindra British Telecom on a wide range of systems. His simulation experience involved using GPSM to simulate commercial software development processes. Mr. Nayak is currently a doctoral student in Portland State University with research interests in software process modeling and simulation. He has an MBA from Portland State University with an emphasis on Management of Innovation and Technology, and a B.E in Electronics and Communications Engineering from Mangalore University in India.

Wayne Wakeland

Dr. Wakeland is an Associate Professor of System Science at Portland State University, where he teaches a suite of courses on computer simulation. His active research areas include software process simulation, biomedical simulation, and optimization in the context of simulation. Wakeland completed his Ph.D. in Systems Science at Portland State University after earning B.S. Engr. and M. Engr. degrees at Harvey Mudd College. His background also includes twenty year of industrial management experience in information technology and manufacturing.

References

[1] Raffo, D.; Spehar, G.; Nayak, U., "Generalized Process Simulation: What, Why and How?", Proceedings of ProSim '03 Workshop, May 2003.

[2] Parnas, D. L. "On the Criteria To Be Used in Decomposing Systems Into Modules", Communications of the ACM, Vol. 15, No. 12, pp. 1053-1058, December 1972.

[3] Parnas, D.L., "On the Design and Development of Program Families'", IEEE Transactions on Software Engineering, Vol. SE2, No. 1, March 1976, pp. 1-9.

[4] Parnas, D. L., "Designing Software for Ease of Extension and Contraction", IEEE Transactions on Software Engineering, Vol. SE-5, No. 2, March 1979, pp. 128-137.

[5] Parnas, D. L., "The Modular Structure of Complex Systems", IEEE Transactions on Software Engineering, Vol. SE-11, No. 3, March 1985, pp. 259-266.

[6] Parnas, D. L., Clements, P.C., "A Rational Design Process: How and Why to Fake It", IEEE Transactions on Software Engineering, Vol. SE-12, No. 2, February 1986, pp. 251-257.

[7] Weiss, D. M.; Lai, C.T.R., "Software Product-Line Engineering: A Family-Based Software Development Process", Addison-Wesley, 1999, 448, pgs., ISBN 0-201-69438-7.

[8] Gardner, K., "Cognitive Patterns: Problem-Solving Frameworks for Object Technology", Addison-Wesley, 1998, 250, pgs., ISBN 0-521-64998-6

[9] Ribó J. M.; Xavier Franch X. A precedence-based approach for proactive control in software process modeling, ACM International Conference Proceeding Series, Proceedings of the 14th international conference on Software engineering and knowledge engineering, Ischia, Italy, 2002.

Simulating Problem Report Flow in an Integration Process

Dan Houston, Ph.D.
Aerospace Systems Software CoE
Honeywell International, Inc.
dxhouston@ieee.org

Abstract
Software process improvement (SPI) has been a discernible theme in the literature on software process simulation. This literature has recognized a wide variety of ways in which simulation can support SPI. This case study describes one of those ways. Very focused, retrospective modeling of integration problem report flows provided insight into integration dynamics. The insight gained, both from modeling the recent development phase and from modeling some alternative scenarios, clarified the lessons learned and suggested a major improvement for the next release cycle.

Keywords: Software process improvement; integration problem processing; integration simulation

1. Simulation and SPI Background

Software process improvement (SPI) has been a discernible theme in the literature on software process simulation. In their scheme for characterizing software process simulation, Kellner *et al.* [1] listed SPI as one of six purposeful categories. Several studies have supported this characterization.

In particular, several studies have related simulation for SPI to industry standards. Christie [2] and Raffo and Vandeville [3] both discussed simulation in the context of CMM-based SPI. Christie suggested a role for simulation in progressing to each CMM level. Raffo and Vandeville described the use of simulation in a company that embraced the CMM as part of its strategic process improvement goals. Stallinger [4] described the development of a model of process capability evolution based on ISO 15504.

In contrast to standards-based SPI deployment of process simulation, some authors report the use of smaller, more focused modeling efforts that influence managerial decision-making. Madachy and Tarbet [5] reported on the use of six small models. Use of two of them in particular, a model of Brooke's Law and a model of personnel switching contexts, produced changes in managerial policies.

Clearly, the literature has recognized a wide variety of ways in which simulation can support SPI. Furthermore, the categories of simulation purpose offered by Kellner *et al.* [1] can be enhanced by suggesting several dimensions for distinguishing simulation-supported SPI.

- *Standards-based vs. business-driven.* An agenda for SPI, and therefore for simulation support, may be guided by the use of an industry standard, or it may be driven directly by needs for improving the business results of programs.
- *Modeling scope.* Madachy and Tarbet distinguished five levels of model scope: a portion of a development life cycle; a development project; multiple, concurrent projects; long-term product evolution; and long-term organization.
- *Retrospective vs. predictive.* Retrospective modeling can be performed either for the sake of measuring savings from improvements, or for the sake of understanding problems and what might have been done differently. It has all the advantages of hindsight, including data for model calibration. Predictive modeling looks ahead and generates new information for decision-makers.

In terms of these categories, this case study describes use of a model based on immediate business need. The scope was a single phase, integration, and the model was primarily retrospective for the sake of understanding the processing of integration problems.

2. Case Study Background

Industrial software development often takes place under intense schedule pressure. In this type of setting, large process improvement projects are difficult to undertake; the time and effort required for simulating a complete development process can be prohibitive. As a result, SPI opportunities must be studied quickly and the best improvements selected and implemented incrementally. Consequently, improvement must often begin by

focusing on a particular phase of concern. Such was the case in an avionics system program.

An avionics system provides numerous functions in areas that include flight planning, crew displays, navigational aids, vertical and lateral guidance, and aircraft system monitoring. These functional subsystems must be integrated within themselves, but must also integrate with other aircraft systems such as flight controls and communications.

The software for a new avionics system is typically developed over a period of one to three years. In developments for a large new commercial aircraft, the system can be delivered incrementally, each increment about 4 to 6 months apart. The functionality for each increment is defined with the customer. Depending on the size of the new system, each increment may be managed as a program consisting of development projects based on product lines. At the procedural level, software development processes can vary across product lines, and therefore from one project to another within a program due to factors such as geographical distribution, automated code generation, software artifact reuse, technical maturity of the team members, team stability, product innovation, and team's domain experience.

The complexity of an avionics system means that the integration phase of development is a major undertaking. In 2003, a study of the integration process found that delays often occurred due to scheduling lab time. The integration process was improved by adding tool support in the form of an application for scheduling lab usage. This application, Integration Test Session Support (ITSS) also captures test session data in the form of integration problem reports (IPRs) stored in a database and routed to the appropriate assignee for investigation and evaluation.

3. Preliminary Data Analysis

The first analysis of ITSS data was performed near the end of the first release cycle during which it was fully deployed. Test session data and IPRs were analyzed in order to better understand the integration activity. The study looked at relationships among test session variables (frequency and duration), builds, problems logged, and testers. The following were among the lessons learned from the original analysis of ITSS data.

- Builds were performed daily and the duration of testing on sequential builds was weakly autocorrelated. That is, the time spent testing a build was correlated with the time spent testing the previous build. This autocorrelation suggested that build stability was a factor in test duration. In other words, the more stable a build was, the more it could be tested. As each build became more stable, more testing could be done, until a very unstable build was produced.
- The number of problems logged for a build was a linear function of the build test duration (R^2=.77).
- The types of problems found varied through the program, with integration problems being found first. The types of problems found in integration shifted as functionality was added and as early problems were cleared.

This study produced a number of recommendations for improvements to the ITTS tool, to build planning and management, and to integration management.

4. Problem Report Process and Model

At the same time that the integration study was undertaken, the entire development process was also being analyzed for opportunities to "lean it out" and improve cycle time. Lean analysis works from a product perspective to find places in the product development in which product sits idle. The wasted times may result from rework, from poor planning, from queuing problems, and so forth. All of these are opportunities for cycle time improvement.

In order to take advantage of the ITSS data study and relate it to the lean analysis, Christie's advice was taken. In his description of progress toward Level 3, he states, "Simulation can be a significant help here in identifying critical process dependencies, weak links and production bottlenecks." [2] The ITSS data provided the ability to trace IPRs from origination to disposition so that the timing could be characterized.

The static data analysis indicated that processing of some classes of problems was a source of delays. The automation provided by ITSS had streamlined the integration process, however, its high degree of usability also facilitated a substantial increase in the number of IPRs that had to be processed. Although the problem report processing is conceptually simple (Figure 1), the numbers of testers (95) and investigators (91) complicated the analysis. Thus, the effect of delays was difficult to appreciate until the process was modeled dynamically.

A simple, discrete event model of problem report processing was produced from the ITSS data analysis. The model, illustrated in Figure 2, used one entity type to represent IPRs generated by integration testers. The analysis had identified ten testers who generated most of the IPRs. These ten were modeled individually and the other testers were aggregated in the model. The IPRs were generated deterministically by using the ITSS data directly. By generating squawks deterministically, the model replicated exactly what happened in the

Fig. 1. Integration Problem Reporting Process

process. It also reduced the burden of model validation and it provided a baseline for evaluating stochastic variations of the model.

Each IPR has a classification and a disposition, which were treated as IPR attributes. Classification designates the functionality in which the problem occurred. It is assigned by the tester at the time an IPR is written. IPRs are routed to investigators based on the Classification attribute.

Disposition is the result of the IPR investigation and is the reason for closing an IPR. It may be "Defect," "Unable to reproduce," "Non-problem," or "Duplicate." Because the modeling was retrospective, the disposition was available for all IPRs that had been closed at the time the model was completed. The model's second decision node read the Disposition attribute and either routed the IPR for closure or for fixing.

The simulation time of the model was in workdays. The model included ITSS data on 3400 IPRs generated over 123 workdays. Though the program actually ran about 133 workdays, the model was sufficient to demonstrate the integration delay problem and alternative cases prior to the end of the program.

The ITSS data analysis identified four investigation queues (IPR Types A, B, C, and D) that were backlogged. However, that analysis could not show how those four queues grew, so they were modeled individually; all other problem investigation queues were aggregated in the model (IPR Type Other).

Due to a limitation in ITSS, IPR closure dates were not recorded. Therefore, IPR investigation durations had to be estimated from other data sources. Using defect records and ITSS data, investigation durations for all IPRs were found to be exponentially distributed, so the investigation durations for each IPR type were assumed to be exponentially distributed.

During modeling, it became apparent that performance of the Type A queue dominated the processing. The Type Other queue, actually an aggregation of 87 small queues, handled most of the IPRs. The Type B, C, and D queues were found to have relatively small backlogs, which occurred only because the people handling them were redirected to support Type A investigations. Consequently, IPR Types B, C, and D have artificially inflated investigation durations and their backlogs could have been cleared quickly.

Given their queue sizes, The Type A and Other investigation durations were modeled as exponential distributions. However, the Type B, C, and D investigation durations were not important, so they were modeled simply as constants.

Once IPRs are investigated, the ones that are not defects ("Unable to reproduce," "Non-problem," or "Duplicate") are routed to closure. The IPRs determined to be defects are routed for fixing only when not covered by an existing defect report. As 48% of the defect IPRs did not already have defect reports, a uniformly distributed random number (U(0,1)) was used to select these entities.

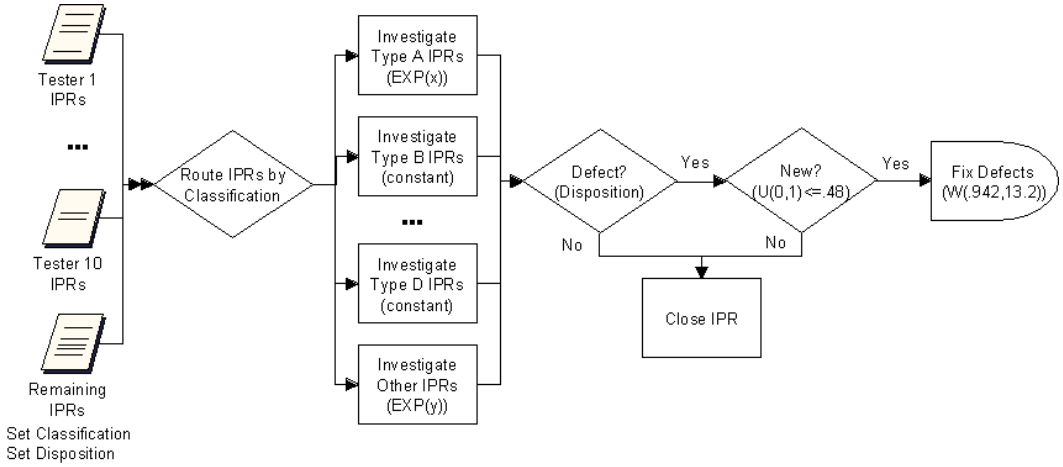


Fig. 2. IPRs Model Diagram

Fix times were distributed Weibull(.942, 13.21), which has a mean of 13.5 workdays. The rework cycle is modeled as multiple server, so entities depart after their assigned delay.

4. Model Usage and Results

4.1 Base Case

The actual results of the integration problem processing were considered the base case. The model allowed us to study the workflow after deployment of ITSS. We found that delays had moved downstream in the integration process. ITSS enabled more integration testing and easier logging and assignment of IPRs. As a result, more IPRs were being generated. Delays due to a lack of coordination in integration testing and manual handling of IPRs were significantly reduced. As a consequence, the delays due to IPR investigation became significant.

Figure 3 shows the number of IPRs opened (top line) and closed (middle line) up until Workday 123. Also, the "All Closed" line (bottom) shows the number of IPRs and defects closed. Therefore the vertical distance between the IPRs Opened and IPRs Closed lines is the number of IPRs awaiting investigation or being investigated. Similarly, the vertical distance between the IPRs Closed and All Closed lines is the number of defects being fixed.

The overall problem generation and disposition rates could be characterized, for the most part, as discontinuous linear functions. Between Workdays 12 and 60, the IPR generation and disposition rates were equal, 24 per day. At Workday 60, the generation rate doubled to 48 per day, but the disposition rate only increased by a multiple of 1.6 to 39 per day. The aggregate lengths grew steadily until Workday 108 when the generation rate began to decrease. The program managers knew from other reports that a backlog of IPRs was being worked, but they didn't have a clear understanding of how it had grown.

Figure 4 shows the queue lengths of the four significant queues, Types A, B, C and D. The Type A IPRs queue aroused the greatest concern because it was the longest single queue. Its length peaked at 138 problems on Workday 99, while the other queue lengths had much lower peaks. But it also was the most expensive queue because it was on the project's critical path [6]. For this reason, the three resources investigating the Type B, C, and D IPRs were redirected to investigating the Type A IPRs. Thus, the services rates and the queue lengths were longer than necessary. It was expected that the Type B, C, and D IPR queues could have been cleared in a short time, given attention by their assigned resources.

The simulation provided a quick translation of queue length to development time by allowing the simulation to run beyond the current date until the queues clear (Figure 5). The Type A IPRs queue had an eleven day backlog, whereas the aggregated queues had only a six day backlog. The Type C and D IPR queues took longer to clear due to their inflated service times.

.

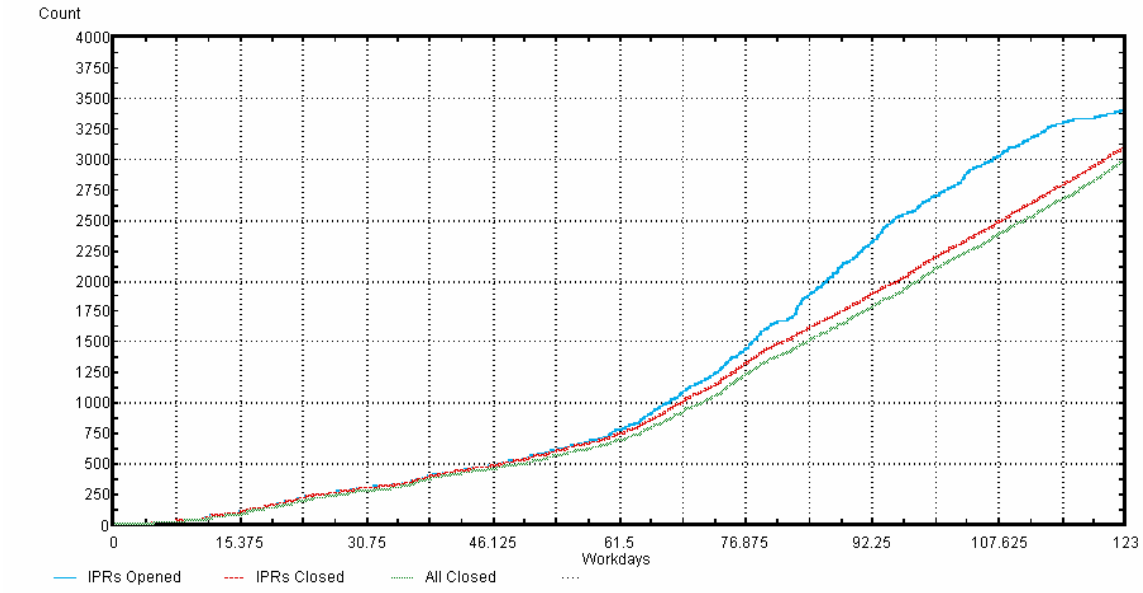


Fig 3. Open IPRs, Closed IPRs, and Closed IPRs and Defects through Workday 123

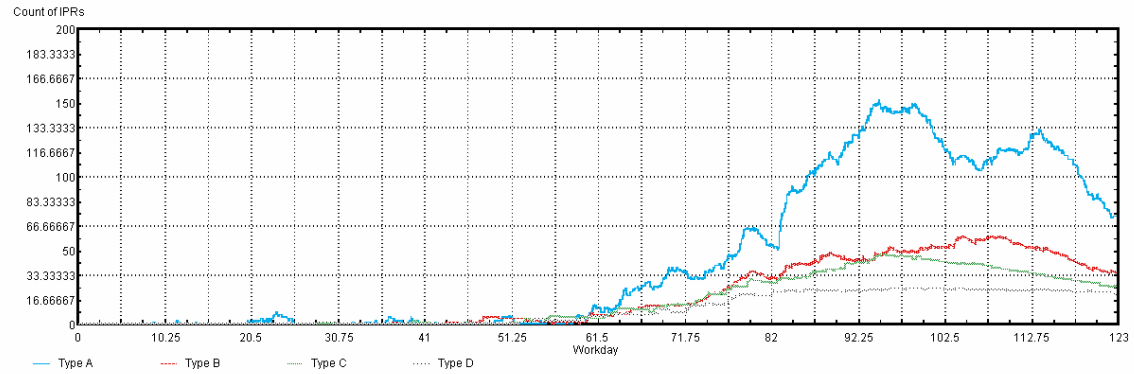


Fig. 4. IPR Queue Lengths: Type A, B, C, and D IPRs

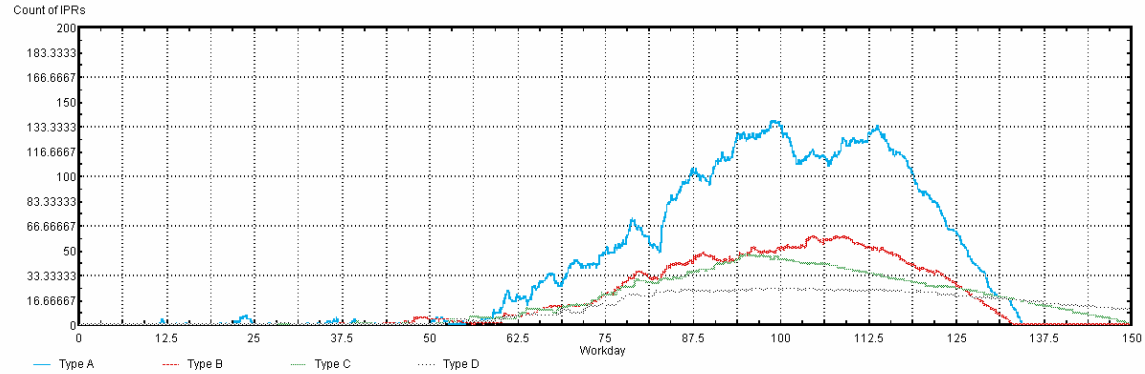


Fig. 5. IPR Queue Lengths Cleared

Although the Type A IPRs queue contained over two weeks of investigation work for about two months, the cost of the queue was recognized as much more than two weeks of cycle time. Reinertsen [6] demonstrates that the cost of a queue increases exponentially with utilization: when utilization is below 65%, the relative time in queue is low, but it doubles from 60% to 80% utilization, from 80% to 90% utilization, and from 90% to 95% utilization. As will be shown, both the time in the system and the variation of the time in system rise dramatically with 100% utilization. Due to the nature of the problems in this queue, some of them were recognized as blocking parts of integration testing. Consequently, the costs of the queue were (a) the measurable delay due a backlog, (b) unpredictability of completion due to high time in system variation, and (c) the unknown delays produced by holding test-blocking IPRs in queue.

4.2. Alternative Cases

A significant advantage of a simulation model is in considering alternative scenarios, that is what might have been. In this case, thinking about alternatives was important because the next release was being planned as the current release was being completed. In this program, the integration process had been improved with ITSS and it was helpful to see whether further improvement was justified, or attention should be shifted to other development phases. Some managers were in favor of additional, easy integration improvement, while others saw greater need to focus more effort on higher quality earlier in the development process. The model afforded an opportunity to look at the relative value of three improvement alternatives, one with regard to higher upstream quality, and two integration improvements

- Better quality entering integration. Though the management team had not settled on a particular project for improving product quality, project ideas were being produced in the course of an assessment during the program. Evidence gathered during the assessment suggested that developers were submitting work for integration prematurely due to schedule pressure. This led management to think that the number of IPRs could be reduced in the next program by better coordination among functional areas and by requiring more cross-functional testing prior to integration. Other quality-inducing improvements could be made, but they could be expensive. Depending on the combination

of actions employed, it was expected that the number of IPRs could be reduced by 25% to 50%. Therefore, these two cases were modeled.

- Better integration testing. The ITSS data analysis had found that 32% of the IPRs were non-problems, duplicates, or could not be reproduced. The analysis had also demonstrated different capabilities among integration testers. Taking these two facts together, managers began to wonder if better testing could eliminate the investigation time due to unnecessary IPRs. If so, then integration testing effectiveness could be improved by having the best testers train and/or mentor other testers. Although elimination of all non-essential IPRs was considered an ideal situation that would be difficult to achieve, it was also worth investigating in a model, so it was modeled.
- More problem resolution capacity. A simple way of decreasing delays was to provide more IPR investigation capacity. This could be done either by increasing the number of investigators, or by using investigators more efficiently. Given resource constraints, the former approach was unlikely; the latter approach was very feasible but required that management have very timely information on investigation queues so that work could be assigned to reduce investigation delays. The case of 25% more investigation capacity for the Type A IPRs was chosen to represent the ability to apply resources immediately as needed to this critical queue.

Table 1 lists the cases modeled, the labels used in the following figures, and a brief description of how the modeling was done. For the alternatives to the base case, the modeling description is a modification to the base model.

Figures 6 and 7 provide comparative results of the cases. Figure 6 is a line plot, one line per case, of the mean IPR time in system. The IPR time in system (TIS) is the duration from an IPR's generation to its disposition. Thus, TIS includes both queue time and investigation time. The lines in Figure 6 show how the mean TIS increased in each case up to Workday 123.

Figure 7 is a similar plot of the standard deviation in IPR TIS at any given time during the program up to Workday 123. Both the mean TIS and the TIS standard deviation remain low in all cases through Workday 60. However, as queue lengths increased, so did the mean TIS and the TIS standard deviation.

These figures show that increasing the IPR investigation effort for Type A IPRs had little effect on both the mean TIS and TIS variation. Better testing ("No Waste") and a 25% IPR reduction produced substantial improvements in mean TIS and TIS variation. The best improvement was produced by a 50% reduction in IPRs.

Prior to this study, managers thought that further improvement to the integration process would be inexpensive and would reduce integration time. However, the results indicated that much more substantial improvements would come from improving the upstream development process so as to improve product quality entering integration. Therefore steps were taken to improve the development cycle.

- Both customer and internal requirements were reviewed earlier so that they could be clarified before they were needed.
- More time was spent studying functional dependencies and planning an incremental development schedule based on these dependencies.
- The large number of problems entering integration produced unstable builds that were difficult to investigate and fix. In order to ensure build stability and satisfaction of build exit criteria, functional increments were planned for 4 weeks apart rather than 2 weeks apart. Build planning included tracing the build plan to requirements.
- In order to ensure readiness for integration, guidelines for integration readiness were revisited. Revised guidelines included focused and rigorous pre-integration testing of developer builds prior to submission for full integration.
- A small change was made to ITSS, the addition of IPR Closed Date. This allowed IPR TIS to be measured directly so that the effects of integration changes could be monitored in subsequent programs.

Table 1. The Cases Modeled

Case	Label	Modeling
Base	Base Case	Replicated actual IPR flow.
Better quality entering integration	25% Less IPRs	25% of IPRs were randomly selected and removed.
	50% Less IPRs	50% of IPRs were randomly selected and removed.
Better integration testing	No Waste	Removed non-problem, duplicate, and non-reproducible IPRs from the flow.
More problem resolution capacity	+25% Invest	Reduced the expected Type A IPR resolution time such that 25% more IPRs could be resolved daily.

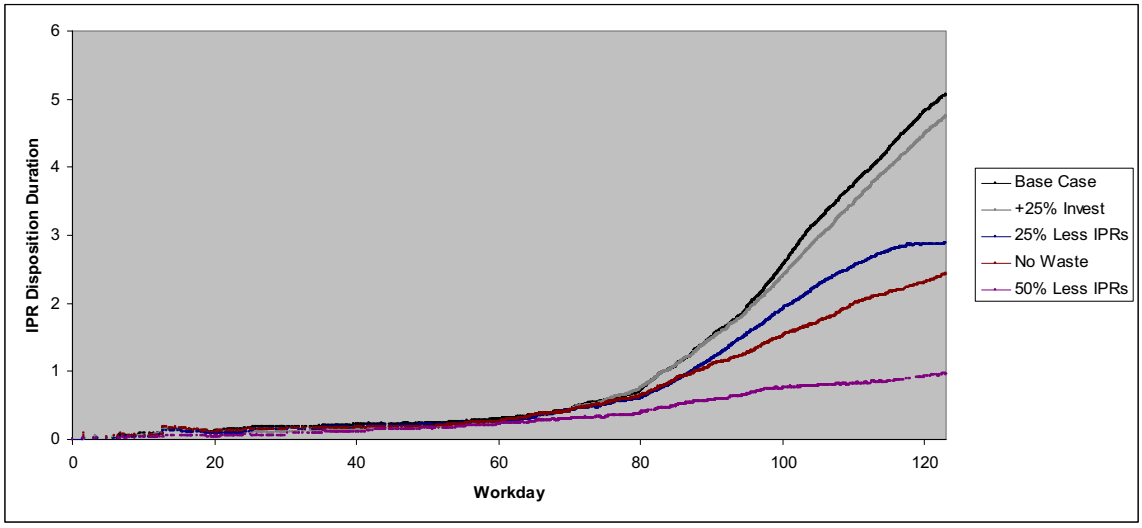


Fig. 6. Mean IPR Disposition Times for Five Cases

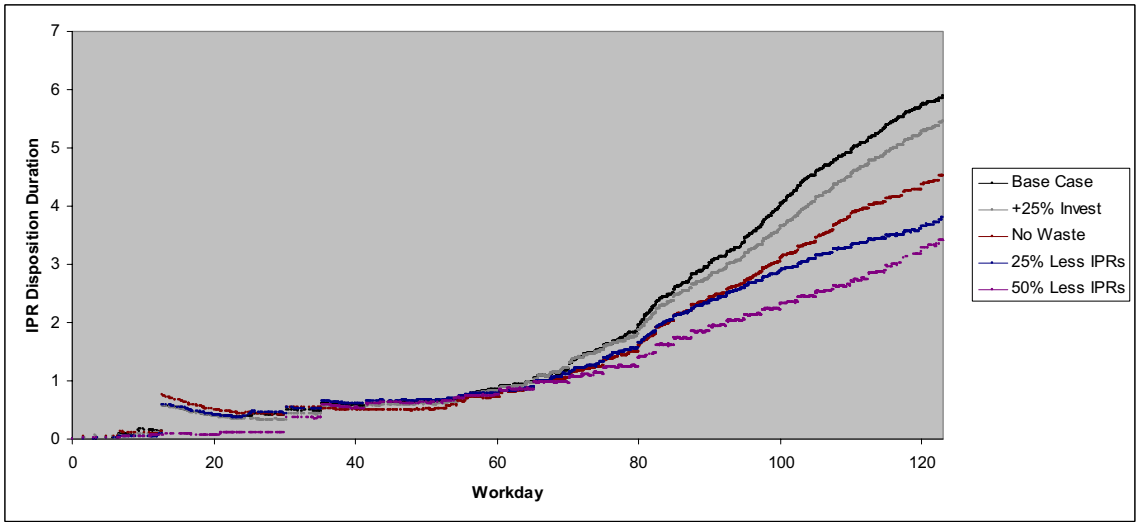


Fig. 7. Standard Deviations of IPR Investigation Times for Five Cases

- The addition of full reporting capability to ITSS was investigated in order to support better integration management and more problem resolution capacity. But due to its low benefit, this work was postponed when it was found that it would cost more than expected.

In addition to the specific program benefits produced by the study, the simulation also contributed to increased "practice pull" [7], that is the active participation of software practitioners in developing and adopting new practices. The lack of a desire for process improvement often limits the effectiveness of SPI initiatives. As individual contributors, developers tend to optimize work practices based on their own job scopes and closest working relationships. A simulation can demonstrate how changes, that might require something more from individuals, benefit a development process.

4.3 Model Verification, Validation, and Evaluation

Richardson and Pugh's [8] framework (Appendix A) was used to verify, validate, and evaluate the model. In addition to the verification and validation exercises performed by the modeler (dimensional consistency, traces, base case reproduction, and so forth), the model was validated and evaluated in discussions with both internal process experts and product line managers.

The model was judged to be a good representation of the problem reporting process, with the exception of the last step for defect fixing. This step was too abstract to describe the projected management of defects beyond the date the data was collected (Workday 123). In order to adequately represent defect report processing, defect management activities such as prioritization would need to be added to the model. In spite of this shortcoming, the model provided new insights to the integration problem report process. Managers realized that their existing reports were neither timely enough nor focused enough to provide them the information required for adequately managing IPS queues.

Development and documentation of the model required about 48 hours of effort over 7 days.

5. Conclusions

The use of a very simple, focused simulation in this project was a low cost way to demonstrate a basic development problem that was difficult to see clearly from existing reports. Also, the modeling enabled quantification of the problems encountered and suggested how much alternative types of changes would be necessary for eliminating integration delays. Finally, it provided a tool for monitoring integration in subsequent releases.

References

[1] Kellner, Marc, Raymond J. Madachy, and David Raffo. "Software process Simulation Modeling: Why? What? How?" *Journal of Systems and Software* 46:2-3 (1999), 91-105.

[2] Christie, Alan. "Simulation in Support of Process Improvement." Software Process Simulation Modeling Workshop, ProSim 98. June 22-24, 1998, Silver Falls, Oregon.

[3] Raffo, David, and Joseph Vandeville. "Software Process Simulation to Achieve Higher CMM Levels." Software Process Simulation Modeling Workshop, ProSim 98. June 22-24, 1998, Silver Falls, Oregon.

[4] Stallinger, Friedrich. "Software Process Simulation to Support ISO/IEC 15504 Based Software Process Improvement." Software Process Simulation Modeling Workshop, ProSim 99. June 27-29, 1999, Silver Falls, Oregon.

[5] Madachy, Ray, and Denton Tarbet. "Initial Experiences in Software Process Modeling." *Software Quality Professional* 2:3 (June 2000) 15-27.

[6] Reinertsen, Donald G. *Managing the Design Factory*. Simon & Schuster, New York, 1997.

[7] Börjesson, Anna, and Lars Mathiassen. "Successful Process Implementation." *IEEE Software* 21:4 (July/August 2004) 36-44.

[8] Richardson, George P., and Alexander L. Pugh III. *Introduction to System Dynamics Modeling with DYNAMO*. Cambridge, Massachusetts: The M.I.T. Press, 1981.

Appendix A

Model verification, validation, and evaluation was performed according to the method of Richardson and Pugh (1981), outlined in this table.

	Structure	Behavior
Verification	Dimensional consistency Extreme conditions in equations Structural adequacy (modeler review)	Parameter variability / sensitivity analysis Structural insensitivity Traces
Validation	Face validity (expert review) Parameter validity	Statistical tests Case study: replication
Evaluation	Appropriateness of model characteristics for the intended audience (user review)	Check for unexpected behavior Check for process insights

Part 4

Position Papers

A Conceptual Model of the Software Development Process

Diana Kirk, Ewan Tempero
Department of Computer Science
University of Auckland
New Zealand
{*d.kirk*|*e.tempero*}*@cs.auckland.ac.nz*

Abstract

We believe that all software development processes can be described by a single abstraction that accounts for the great variety of processes that exist, outcomes of interest and project-specific factors that occur in real-world projects. The abstraction is best represented as a conceptual model, the components of which are theoretical constructs, or models, in their own right. We believe that such a model will facilitate model building to solve individual problems, provide support for empirical research and, in the long term, provide a tool for predictions based on understanding. We are taking the first steps in building such a model.

1 Introduction

An ideal predictive model of the software development process would encompass all kinds of processes, outcomes of interest, and variations in project-specific factors. Such a model would support researchers by providing a holistic framework within which research findings might be positioned and assumptions exposed. Current predictive models are scoped to solve a particular problem on a particular process. These models use existing data as a basis for predicting outcomes, and are thus aimed at understanding and predicting within the bounds imposed by that data. These factors make it difficult to generalise to other problems and processes. If we are to improve our ability to predict, we must first understand in a theoretical way the relationships between process, product and people. Building models (theories), generating hypotheses from these and applying empirical research to test the hypotheses is an accepted approach.

It is our position that all software development processes can be described by a single abstraction and that such an abstraction will facilitate model building to solve individual problems, provide support for empirical research and, in the long term, provide a tool for predictions based on understanding.

The rest of this paper is organised as follows. In section 2 we use a brief history of the study of planetary motion to illustrate the difference between models that just provide prediction, those that provide fragmented understanding and those that provide holistic understanding. In section 3 we overview the kinds of work researchers are doing and examine limitations. In section 4 we suggest what a conceptual model should provide to researchers and present an overview of our model along wih a research roadmap. In section 5 we summarise the paper.

2 Science and Understanding

Rivett [18], when describing the status of model building in the field of operations research in 1972, reminds us that, throughout history, man has constantly searched for pattern and generalisation. From around 700 BC, the Babylonians measured and recorded the motions of the stars and planets, analysed these, and were successful in forecasting planetary events with great precision. Their recordings of hundreds of years of planetary data enabled them to estimate the value of the motion of the sun from the node with an error of only five seconds. The same estimation, when made in the nineteenth century, yielded an accuracy of only seven seconds.

Although the Babylonians recorded events with care, they made no attempt to theorise. The Greeks, on the other hand, followed a different approach, and built first mechanical and then geometrical models of planetary motion in an attempt to understand and explain. However, their models were made up of a number of parts and the Greeks had no success in unifying these. When applied to the Babylonian data, the models were found to be incorrect [18].

Rivett notes that, in more recent times, Kepler proposed three laws of planetary motion based on data that had been collected by Tycho Brahe. He applied an elliptical model to the motion of the planets and from this model produced laws that appeared to work. No-one knew the fundamen-

tal reason why the laws worked. And we notice that, as the laws were based on planetary data, these laws could not predict the movements of other celestial objects, for example, comets. Newton later brought some understanding to bear on celestial motion when he postulated a force that acted between all objects in the universe with mass. From this understanding and unification of ideas from physics and astronomy, he was able to show that orbits for celestial objects, for example comets, were not only elliptical, but could be hyperbolic and parabolic. He was thus able to predict accurately for all celestial bodies, show that Kepler's Laws were a special case of Newton's Laws and improve the accuracy of Kepler's calculations.

Rivett summarises by stating that a model may be predictive without being explanatory, but an explanatory model is always predictive. We also note that an inability to unify fragmented models that relate to the same phenomenon is an indication that understanding is not complete.

A theory is a model or framework for understanding. Building theoretical models, generating hypotheses from these and applying empirical research to test the hypotheses is an accepted approach to better understand cause and effect relationships.

3 Current Models

If we consider the research currently taking place in the area of software process modelling, we can identify a situation similar to that described in the previous section. Models built by researchers are based on specific data sets and are thus predictive but not explanatory, and a number of different modelling paradigms are used. There are three main research groups and each carries out research from a different perspective.

3.1 Simulation Modelling

Simulation models aim to solve a number of different problems, many of which are predictive in nature. Different modelling paradigms are used, for example, discrete-event and system dynamics, and modellers approach a problem from the viewpoint of the paradigm selected. Models are generally created for a specific company process and tend to use metrics data from the target company for model formulation. The scope of application of the models is thus limited and cost of model creation high. Human factors are often included as part of the model architecture, for example, as changing engineer motivation during long projects. However, some human aspects are often 'hidden' in the metrics that populate the model. For example, a company's metrics database may contain a measure of 'typical productivity' or 'average number of defects injected or found', and these metrics in fact 'hide' the fact that real people are coding at a certain rate and certain level of proficiency. If we don't know what were the human factors at play when the metrics were collected, we have no idea whether or not we may apply the same metrics in another project.

3.2 Predictive modelling

Early predictive models aimed to facilitate the prediction of costs and durations for a given project. These cost estimation models are equations linking costs to the size of the software product to be delivered and a number of other factors believed to influence costs, for example, developer experience. The form of the model equation is inferred from a statistical manipulation on a number of datasets collected from real projects. A more recent example of the use of statistical techniques on large datasets is the attempt to isolate factors that affect fault distribution in code [2, 7, 15, 9, 14].

The statistical literature warns us that in this modelling paradigm we must be wary of confusing predictive capability with causation, and that we may apply predictions with confidence only in circumstances similar to those present during original data capture [5].

3.3 Empirical Research

The last few years have seen the emergence of an interest in sound empirical research. It is generally agreed by researchers that, if we are to progress as a professional discipline, it is now time to move away from the 'analytical advocacy research' [6] with which the industry is familiar and towards a more formal approach to experimentation.

Research into the software process can be categorised as examining the inter-relationships between process, product and people. For example, "Which inspection technique is better?", "Did the technique yield better results if the developers were experienced?".

As pointed out by Carver et al. [4], there is a problem with understanding what are the common assumptions arising in current empirical research efforts. It is difficult for researchers to be sure that all possible explanations for results have been identified and that effects are, in fact, due to the cause under investigation. This is a problem of internal validity. For example, in experiments involving process and product, are we certain the human factors were held constant? We remember to take account of experience and skills, but are there any other factors that might confound results, for example, motivation and ease of communications?

3.4 Need for Holism

The limitations exhibited by the above research paradigms can be summarised as relating to context, in that

the full context for the data supporting the model must be known and results obtained are not applicable to different contexts.

We suggest that these limitations might be mitigated by the existence of a holistic framework within which to position research efforts. This viewpoint is supported by a number of researchers from both within and without the software engineering community, who reiterate the need to build models to increase understanding [1, 8, 10, 13, 18].

4 Conceptual Model

Our research is aimed at developing a conceptual model for the software development process. The model captures aspects of people, process and product into a framework that will facilitate a theoretical approach to software development process research. In this section, we overview the current status of our research.

4.1 Objectives

Our model should provide the following support to researchers [12]

- Choice of granularity in the process under study (the whole process, a single process element, etc.).
- The ability to select process elements and to define new kinds of elements.
- A means of capturing the deliverables-related objectives that are the targets for the process.
- A choice of models for the above objectives, for example, quality models.
- The ability to define new kinds of product-related objectives, for example, business value.
- Decoupling of human- and techology-related factors.

4.2 Architecture

In figure 1, we present an overview of the architecture for our model, presented as a UML diagram. The main idea is that any software development process involves a *Product* being changed in some way by application of a number of *Activities*. Each Activity has two dimensions, a *Method* that defines how the Product changes and a scale factor, *EngineerMultiplier*, that captures how the change is scaled according to human related factors. Each model component is a model in its own right. An early version of the model is described in [11]. We highlight some key aspects of the model below.

- A Product comprises all artifacts that describe the software being produced. These include the software delivered to the end customer and all requirements, designs, code, etc. to be delivered to the development organisation as an asset for use in later projects.

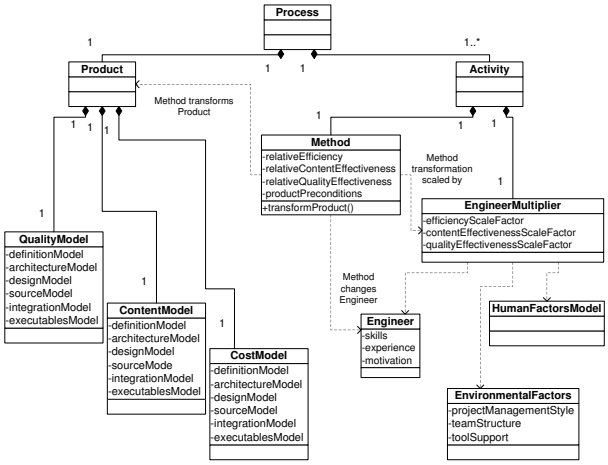


Figure 1. Process model architecture

- We note that any Activity comprises two parts, Method (the task that is carried out) and EngineerMultiplier (how well the Engineers carry out the task).
- A Method is defined by how it changes the Product and what are its Product-related preconditions (for example, existence of design documents). This means that, in addition to the traditional tasks, for example, 'code from design documents', our model handles any task that causes change to the Product. Tasks such as 'test first design', 'create a prototype based on a feature list' or 'code from prototype' are valid Methods.
- The definition of Method permits tasks of any granularity. So, for example, 'develop product from requirements' or even 'develop product' are as valid as 'carry out design review'.
- When an Engineer carries out a Method, the Engineer is changed as a result. For example, his experience or some skills are likely to increase.
- A Product comprises a number of Product models. At the moment these relate to the conventional product-related drivers i.e. *ContentModel* ('how much is there'), *QualityModel* ('how good is it') and *CostModel* ('how much did it cost'). However, as we become aware of other objectives, for example, the need to capture business value for product artifacts[3], we can simply extend the model to encompass these.
- In our model, we view environmental factors as affecting how well the Engineers carry out tasks. A HumanFactorsModel defines what are the relevant factors and how these are combined into our EngineerMultiplier.

4.3 Implementation

Our first implementation of the model uses java, with product and process definitions in an input .xml file. This implementation was selected for convenience.

4.4 Proof of Concept

As preliminary proof-of-concept feasibility study for our model, we have captured two simulation modelling examples from the literature and are in the process of capturing a third. Note that we are not re-creating the research, but are aiming to confirm that we can *represent* a range of experiments with our model. We present an overview of these examples below.

For each study, we first define the appropriate Product Model, then identify the Methods required for Product transformation, and finally identify and calculate values for the Efficiency and Effectiveness multipliers that represent both relative Method efficacy and how well the Engineers work with the Method.

4.4.1 Example 1

This proof-of-concept feasibility study is based on work by Raffo, Vandeville and Martin [17]. This paper describes a simulation model to examine quality outcomes when the review and inspection processes are improved. A state-based model is implemented.

For this experiment, the Product Model includes 'Number of remaining defects' and 'Number of discovered defects'. The Methods identified are listed in the first column of table 1. The relevant efficacy factor is 'QualityEffectiveness' and the values obtained are summarised in table 1 for the 'as-is' process and the 'to-be' process.

Table 1. Example 1 efficacy summary

Activity	As-is	To-be
Analyse	1.01	1.01
InspectArchitecture	.276	.65
ResolveArchitectureDefects	1	1
DetailDesign	1.003	1.003
InspectDesign	.278	.66
ResolveDesignDefects	1	1
CodeAndUnitTest	1	1
CodeWalkthrough	.28	.66
ResolveCodeDefectsWalkthrough	.91	.976
UnitTest	.63	.627
ResolveCodeDefectsUnitTest	.95	.867
IntegrationTest	.67	.607
ResolveIntegrationDefects	1	1

We note that the values for 'InspectArchitecture', 'InspectDesign' and 'CodeWalkthrough' have increased as expected (the original experiment was to 'improve' these activities).

4.4.2 Example 2

The next study is based on work by Pfahl and Lebsanft [16]. The cited paper describes a simulation model to analyse how much effort would be required to stabilise requirements such as to achieve the most cost-effective outcome. A system dynamic model is implemented. Six experiments take place, each with a different value for requirements effort and the results on total effort studied.

For this study, the ProductModel includes 'total number of requirements', 'number of correct requirements' and 'cost in person hours'. We identify two Methods ('capture requirements' and 'implement requirements') carried out for each release cycle and implement these an appropriate number of times.

The efficacy values of interest are 'Efficiency' and 'ContentEffectiveness' (how many were correct). As the experiments assume constant efficacy values for 'implement requirements', we show the values for the 'capture requirements' Activity only for the six experiments described in the paper (table 2).

Table 2. Example 2 efficacy summary

Simulation	Efficiency	ContentEffectiveness
n1	2.806	.47
Baseline	1	.733
n3	.586	.832
n5	.262	.935
Optimal	.1814	.9615
n6	.151	.9709

The assumption is that more time spent results in better requirements. In the context of our model, we note that as time spent in gathering requirements increases (n1 to n6), efficiency decreases (more time is taken per requirement) and effectiveness increases (they get more correct).

4.4.3 Contribution

In the two examples above, we represented two experiments in our model. In experiment 2, the assumption made by Pfahl and Lebsanft is that spending more time on requirements gathering results in better requirements. When we capture the experiment in our model, the spending of more time is represented as a decrease in Efficiency and an increase in Effectiveness. Moreover, in the context of our model, we understand that efficiency and effectiveness may

be changed either by changing the Method or by changing how well the Engineers work with the Method. We now have a much larger scope for a business case. Rather than a single solution 'spend more time on requirements', alternatives can be proposed to optimise efficiency and effectiveness by strategies of Method fine-tuning, training, personnel skills, etc. A similar oservation results from experiment 1.

4.5 Research Roadmap

The first task is to continue retrofitting experiments from the various research groups into our model, aiming for as much variation as possible, for example, experiments based on different kinds of process and experiments in which process granularity varies from fine-grained to those in which the whole development effort is treated as a single Activity. In parallel with the above, we will investigate what is an appropriate representation for our model. We are considering alternatives, for example, Alloy, to replace the existing UML representation.

Once we are happy with both model and representation, the model will be used for hypothesis generation and formal experimentation. For example, one hypothesis would be that a single HumanFactors model is appropriate for all tasks. The form of the model might then be researched within the academic environment and later, once it is clear what might be the relevant factors, 'tested' in industry.

5 Summary

It is our position that all software development processes can be described by a single abstraction that accounts for the great variety of processes that exist, outcomes of interest and project-specific factors that occur in real-world projects. We believe that such a model will facilitate model building to solve individual problems, provide support for empirical research and, in the long term, provide a tool for predictions based on understanding.

Current software development process models predict without understanding and research efforts are fragmented. We hypothesise that an integrated model of the software development process is possible and that such a model would support researchers by providing a holistic framework within which research findings might be positioned and assumptions exposed. We present a candidate for such a model and some initial proof-of-concept studies.

We are aware that this is an ambitious undertaking. There are many reasons why such a model can not be contructed in entirety at the present time. For example, we are not confident about how to best operationalise and measure many software attributes, the industry is not agreed on what is an appropriate quality model, etc. This is a task for many researchers over a period of time. A collaborative approach is indicated.

References

[1] F. Anger. Directions for the NSF software engineering and languages program. *Software Engineering Notes*, 24(4), 1999.

[2] V. R. Basili and B. T. Perricone. Software errors and complexity: An empirical investigation. *Communications of the ACM*, 27(1), 1984.

[3] B. Boehm. Value-based software engineering. *ACM SIGSOFT Software Engineering Notes*, 28(2), 2003.

[4] J. Carver, J. V. Voorhis, and V. Basili. Understanding the impact of assumptions on experimental validity. In *Proceedings of the 2004 International Symposium on Empirical Software Engineering*. The Institute of Electrical and Electronic Engineers, Inc., 2004.

[5] T. Coladarci, C. D. Cobb, E. W. Minium, and R. C. Clarke. *Fundamentals of Statistical Reasonong in Education*. John Wiley and Sons, Ltd, USA, 2004.

[6] N. Fenton, S. L. Pfleeger, and R. Glass. Science and substance: A challenge to engineers. *IEEE Software*, July, 1994.

[7] N. E. Fenton and N. Ohlsson. Quantitave analysis od faults and failures in a complex software system. *IEEE Transactions on Software Engineering*, 26(8), 2000.

[8] A. Fuggetta. Rethinking the modes of software engineering research. *Journal of Systems and Software*, 46(2/3), 1999.

[9] L. Hatton. Reexamining the fault density-component size connection. *IEEE Software*, March/April, 1997.

[10] C. Jones. *Programming Productivity*. McGraw-Hill, Inc, 1986.

[11] D. Kirk and E. Tempero. A flexible software process model. Technical Report UoA-SE-2004-3, University of Auckland, 2004.

[12] D. Kirk and E. Tempero. Proposal for a flexible software process model. In *ProSim 04*, 2004.

[13] J. Ludewig. Models in software engineering - an introduction. *Software and Systems Modeling*, 2(1), 2003.

[14] J. C. Munson and T. M. Khoshgoftaar. The detection of fault-prone programs. *IEEE Transactions on Software Engineering*, 18(5), 1992.

[15] T. J. Ostrand and E. J. Weyuker. The distribution of faults in a large industrial software system. In G. S. Avrunin and G. Rothermel, editors, *Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis*, pages 56–64. ACM, 2002.

[16] D. Pfahl and K. Lebsanft. Using simulation to analyse the impact of software requirement volatility on project performance. *Information and Software Technology*, 42, 2000.

[17] D. M. Raffo, J. V. Vandeville, and R. H. Martin. Software process simulation to achieve higher cmm levels. *Journal of Systems and Software*, 46(2/3), 1999.

[18] P. Rivett. *Principles of Model Building: The Construction of Models for Decision Analysis*. John Wiley and Sons, Ltd, Bath, Great Britain, 1972.

People Applications in Software Process Modeling and Simulation

Ray Madachy, *Senior Member, IEEE*

***Abstract*—Focusing on people is one of the highest leverage opportunities to improve process performance. But people issues have been under-represented in process modeling and simulation, as much of the work has dealt with technical aspects. Software processes will always be socio-technical by nature, and due attention should be paid to the human aspects and people dynamics. Simulation using system dynamics is useful to represent and understand some people issues. This position paper describes basic phenomenology and sample simulation results for people considerations. Constructs are discussed for important aspects including motivation, exhaustion, experience and learning curves, training, hiring and retention, communication, stakeholder collaboration, and workforce dynamics at the project and macro levels. A short summary of lessons learned about people issues is also provided.**

***Index Terms*—People factors, Software process modeling, Software process simulation, System dynamics**

I. Introduction and Background

People rather than technology provide the best chance to improve processes and execute successful projects. Software processes will always be socio-technical by nature, and due attention should be paid to the human aspects. Too many proposed "silver bullets" for software engineering focus on technical solutions while ignoring the people dynamics. People must be available, they should have requisite skills and experience for their jobs (including how to work with others), they should be adequately motivated to perform, have necessary resources, and have a good working environment that fosters creativity and learning.

The focus of this work is on phenomena directly related to people. It will cover some important personnel attributes, skill development, training, hiring and retention, team issues, and macro workforce dynamics. Even though many of the people attributes are considered "soft factors", it is still feasible and meaningful to model them [1].

Simulation can also be used directly for training personnel in a project flight simulation mode. However training is a different class of application and not addressed in this paper.

Ray Madachy is with the University of Southern California Center for Software Engineering, University of Southern California, Los Angeles, CA 90089 USA and Cost Xpert Group, San Diego, CA 92109 USA (e-mail: madachy@usc.edu).

A. An Application Taxonomy

People applications are differentiated from other applications in software process modeling and simulation in terms of their primary state variables. In system dynamics models, these correspond to the levels (accumulations) in the main chains of primary focus. People applications quantify personnel levels and characteristics. Process and product applications are centered around software artifacts, their transformations and attributes through the process stages. Project and organization applications generally revolve around issues such as business case analysis, estimating, planning, statusing, resource allocation, tracking business value, etc. Decision structures are frequently embodied in the applications regarding people, processes and products, etc.

There are however many connections and overlap between the application areas. For example, modeling the allocation of resources for personnel development would be an organizational application. This paper discusses only applications where the primary focus is on people and their attributes.

B. People Impacts

How important are people characteristics relative to other factors? Our COCOMO II research [2] provides quantitative relationships for major software project factors. The combined personnel factors provide the widest range of software productivity impact against all other cost factors according to the data. The total productivity range for the people factors is 20.8, which denotes a 2008% variation in productivity due to people attributes. Many of the other factors are also influenced by human actions, so the overall variation due to people is actually greater.

However, COCOMO assumes that people are reasonably well-motivated and that other soft factors can be treated as invariant. These assumptions do not hold true in too many instances, so some people factors not currently represented in COCOMO will be addressed. Most of these represent measures that are harder to define and collect in the COCOMO context, so this work shows some ways they can start being quantified.

II. Applications

A few sample applications will be overviewed. Additional

aspects and more extensive modeling details are provided in [1]. In the first section below we start with modeling the number and kinds of people on a job, and then address other important characteristics.

A. Project Workforce Modeling

Workforce modeling contains structures for hiring, assimilation, training, and transferring of people off a project. Separate levels can be used to represent different experience pools. Frequently there are levels for new and experienced people, since there are productivity differences between the two. Having separate levels also allows one to model experienced people being involved in training of new hires. The training overhead on the part of the experienced people gives them less time to devote to other development tasks. A Brooks' Law model in [1] describes this.

In the Abdel-Hamid integrated project model [3], there was also a work force level needed parameter for the decision on how many people are currently required for the project. The level is limited partly on the ability to assimilate new people onto the project. There is also a work force level sought parameter, which is used in conjunction with the work force level needed parameter. The gap between total work force and work force level sought is used to control hiring and transfer rates.

Acquiring good personnel is often very difficult, but is time well-spent. Personnel must first be found and identified. Then they have to be sold on a particular job, and negotiations ensue until all parties reach a hiring agreement. The time needed to hire new employees depends on the position levels, organizational culture and processes, current market trends etc.

A system dynamics delay structure is a convenient way to model hiring delays. The delay time refers to the period from when a position is open, normally through a job requisition process, to when a new hire starts work. Representative hiring delays are listed and discussed in [1].

Besides structures for modeling the workforce dynamics, there should also be provisions for human characteristics that impact productivity. Some of these important factors are reviewed next.

B. Motivation and Overtime

Motivation can have tremendously varying impact over time. It may even eclipse the effects of skill and experience on projects. Motivation may wax and wane due to a multitude of factors and impact productivity either way. People will traverse through several up or down motivational phases during a project.

Pressure to work overtime can greatly influence motivation and overall productivity. Suppose that management pushes for overtime in order to catch up on schedule, and that the management edict can be expressed in terms of desired output to normal output. We wish to model an effective overtime multiplier for productivity. How should this relationship be modeled?

Limits exist such that people will not continue to increase their overtime past their saturation thresholds. Thus, there is a value of desired output beyond which no further overtime will be worked. Studies have indicated that a small amount of aggressive scheduling will tend to motivate people to produce more, but highly unreasonable goals will have the opposite effect. The management challenge is to strike the right balance.

A few people might work 80 hour weeks and even greater, but a more reasonable limit might be a 60-hour workweek for most industry segments. This puts an upper bound of 60/40=1.5 on the overtime factor. Additionally, each worker will not do overtime to the same extent and may start/stop their overtime periods at different times. Thus, we will model the response of the aggregate system as a smooth curve. Such functions are best done in terms of normalized variables, so the independent variable will be the desired/normal output. A relationship has been created that relates the productivity overtime multiplier as a function of desired/normal output [1]. This function is easily used in a system dynamics model in graphical or tabular form.

In reality different classes of organizations exhibit varying productivity multiplier curves. For example, people in an entrepreneurial startup situation are going to be much more willing to put in overtime as opposed to a civil servant position. Additionally the curves level off when extrapolated further into the over-saturated zone, as they eventually show degraded productivity as the expectation is set too high. For many individuals, the overtime productivity curve will exhibit a downward trend after a point. See [1] for these extensions and refinements to the curve for selected industries.

C. Exhaustion and Burnout

Human nature dictates that the increased productivity effects of overtime can only be temporary, because everyone needs to "de-exhaust" at some point. This has been observed across all industries. People start working a little harder with some schedule pressure, but after several weeks fatigue sets in and productivity drops dramatically. Then there is a recovery period where people insert their own slack time until they're ready to work at a normal rate again.

The underlying assumptions of the exhaustion model from [3] are:

- Workers increase their effective hours by decreasing slack time or working overtime.
- The maximum shortage that can be handled varies.
- Workers are less willing to work hard if deadline pressures persist for a long time.
- The overwork duration threshold increases or decreases as people become more or less exhausted.
- The exhaustion level also increases with overwork.
- The multiplier for exhaustion level is 1 when people work full 8 hour days, and goes over 1 with overtime. The

exhaustion increases at a greater rate in overtime mode up to the maximum tolerable exhaustion.

- The exhaustion level slowly decreases when the threshold is reached or deadline pressures stop with an exhaustion depletion delay.
- During this time, workers don't go into overwork mode again until the exhaustion level is fully depleted.

A run of the exhaustion model with specific parameters described in [1] shows increasing exhaustion up to a breaking point. The actual fraction of man-days for the project rises as the project falls behind schedule, which means that people have less and less slack time during their day. The fraction increases as the work rate adjustment kicks in. As the exhaustion level accumulates, it affects the overwork duration threshold such that the number of days people will work overtime starts to decrease.

These general trends continue and productivity is increased nearing the point of maximum exhaustion. When the overwork duration threshold reaches zero the team cannot continue at the same pace, so the de-exhausting cycle starts. The actual fraction of man-days for the project and the exhaustion level both slowly decrease. The overwork duration threshold begins to increase again, but a new overwork cycle won't start until the exhaustion level reach zero. Alternative formulations for burnout dynamics are also discussed in [1].

D. Learning

Learning is the act of acquiring skill or knowledge through study, instruction or experience. In the context of a software process, developers become more productive over the long term due to their accumulated experience. The increase in productivity occurs indefinitely, and a *learning curve* describes the pattern of improvement over time.

A learning curve is expressed by an S-shaped productivity function over time. The curve shows an initial period of slow learning where the development process is being learned, then a middle period of fast learning followed by a decreasing slope portion that nearly levels out (the form of a typical S-curve). The reduced learning period is due to machine limitations. Calibration of the curve for software development is treated in [1].

The implications of learning while executing software processes can be substantial when trying to plan effort and staffing profiles. Unfortunately, few planning techniques used in software development account for learning over the duration of a project. The standard usage of COCOMO for example assumes static values for experience factors on a project.

At first glance, learning curves may be easily expressed as a graph over time. But this simple depiction will only be accurate when time correlates with cumulative output. As Raccoon discusses [4], there are biases that mask learning curves. There are also process disruptions for periods of time that affect productivity.

Learning curves are traditionally formulated in terms of the unit costs of production. The most widely used representation for learning curves is called the Log-Linear learning curve expressed by $y=a(x)^n$, where a is the cost of the first unit, x is the cumulative output, and n is the learning curve slope. The learning curve formulas are implemented in system dynamics feedback structures and discussed further in [1].

E. Communication

Good communication between people is necessary for project success. Frequently it is cited as the most important factor. Personnel capability factors inherently account for team dynamics rather than assessing individual capabilities. If a team doesn't communicate well then it doesn't matter how smart and capable the constituents are. However, a great deal of time is often consumed in communication.

As shown in a simple Brooks' Law model [1], team size is an important determinant of communication overhead and associated delays. Communication overhead depends on the number of people. The number of communication paths increases proportionally to the (number of people)^2. The model shows that adding people increases communication overhead and may slow the project down more. It takes more effort and longer time to disseminate information on large teams compared to small teams. This could have a profound impact on project startup, since the vision takes much longer to spread on a large team.

As the overall team size increases however, team partitioning takes place and communication overhead is reduced on a local scale. Partitioning is not trivial to model and the communication overhead should be reformulated to account for it [1].

F. Stakeholder Negotiation and Collaboration

Achieving early agreements between project stakeholders is of tantamount importance. Reaching a common vision between people and identifying their respective win conditions are major precepts of the WinWin process [6]. Negotiation and collaboration processes can be difficult though, and modeling the social aspects can provide insight. A couple unique models have explored both the social and organizational issues of requirements development.

An important work in this area is [7] that simulates the organizational and social dynamics of a software requirements development process. It looks at how the effectiveness of people interactions affects the resulting quality and timeliness of the output. The model has both continuous and discrete modeling components.

While the modeling of the organizational processes uses a discrete, event based approach, the social model was based on continuous simulation. The social model gives each individual three characteristics: their technical capability, their ability to win other team members to their point of view (influence), and the degree to which they are open to considering the ideas

of others (openness).

The model showed that the level of commitment or trust escalates (or decreases) in the context of a software project. Commitment towards a course of action does not happen all of a sudden but builds over a period of time.

Another effort using system dynamics to model stakeholder collaboration activities was performed in [8]. They simulated the Easy WinWin requirements negotiation process to assess issues associated with the social and behavioral aspects, and to explore how the outcome is impacted. They revisited [7] and retained the model for individual characteristics of technical ability, ability to influence others and openness to influence from others.

G. *Macro Workforce Shortage*

There is a seemingly ever-present shortage of skilled software and IT personnel to meet market needs. Closing the gap entails long-term solutions for university education, ongoing skill development in industry and worker incentives. Rubin studied the IT workforce shortage in the United States [5] and mentions a destructive feedback loop whereby labor shortages cause further deterioration in the market.

The degenerative feedback is a reinforcing process between worker shortages, staff turnover and job retention. Organizations try to alleviate the labor shortage by offering salary increases to attract new hires. As more people job hop for better compensation, the shortage problem is exacerbated because more attrition and shorter job tenures cause lessened productivity. Thus the market response adds fuel to the destructive feedback loop. The increased compensation to reduce the impact of attrition causes more attrition.

The feedback results in a greater need for people to develop a fixed amount of software, in addition to the overall increased demand for software development in the world market. Considering the larger world picture illustrates interactions between national and global economic trends.

A causal loop structure of the labor shortage feedback loop has been created to describe the situation. The difference between desired productivity and actual productivity (on a local or aggregate level) manifests itself as a workforce shortage. The shortage causes aggressive hiring such as providing large starting salaries (or other hiring incentives) to fill the gap. The increased compensation available to workers causes higher attrition rates. The increased personnel turnover causes learning overhead for new people to come up to speed and training overhead on the part of existing employees to train them. Both of these impact productivity negatively, which further causes the workforce shortage.

III. CONCLUSIONS

People are the single most important aspect to focus on to improve productivity. More will be gained by concentrating on people factors and human relations than technical methods. In this sense agile approaches have it right. The effectiveness of many processes and ultimate project success are dependent on people working together.

Motivation is a key factor to achieve high productivity from people. It can enable people to work very hard and be creative. Yet people have limits on their amount of hard work and long hours until burnout inevitably occurs. The right balance has to be struck. Management needs to be very careful about overtime and its converse slacktime.

Learning is an S-shaped phenomenon that can be successfully modeled, yet plans usually don't account for individual and team learning curves. It is a very real consideration that can be easily handled with system dynamics.

Good communication between people is crucial, both within a development team and with external stakeholders. Individual capabilities aren't as important as communicating well as a team. Emphasis should be placed on creating an environment for teams to gel. Team dynamics play a particularly critical role when identifying and negotiating stakeholder win conditions. For external stakeholders, expectations management is critical to make everyone a winner. It won't work without open, honest communication and negotiation of win conditions to reach a shared vision between people.

Workforce issues should be dealt with at the macro level also. There are feedback dynamics that create and sustain a shortage of skilled people. Without a long-term vision to address the causes then the industry will continue to be plagued with staffing problems. Modeling and simulation can be used to help understand this and other workforce issues.

These are just some of the people phenomena that should be better understood and accounted for on software projects. See [1] for other applications (including using simulation for personnel training), lessons learned, comparisons of different modeling techniques for people considerations and directions for future work.

REFERENCES

[1] R. Madachy, *Software Process Dynamics*, IEEE Computer Society Press, Washington D.C., 2005 (to be published)

[2] B. Boehm, C. Abts, W. Brown, S. Chulani, B. Clark, E. Horowitz, R. Madachy, D. Reifer, B. Steece, *Software Cost Estimation with COCOMO II*, Prentice-Hall, 2000

[3] T. Abdel-Hamid, S. Madnick, *Software Project Dynamics,* Englewood Cliffs, NJ, Prentice-Hall, 1991

[4] L. Raccoon, "A learning curve primer for software engineers", *Software Engineering Notes*, ACM Sigsoft, January 1996

[5] H. Rubin, "The United States IT Workforce Shortage (Version 3.0)", META Research Report, 1997

[6] B. Boehm, A. Egyed, J. Kwan, D. Port, A. Shah, R. Madachy, "Using the WinWin spiral model: A case study", *IEEE Computer*, July 1998.

[7] A. Christie, M. Staley, "Organizational and social simulation of a software requirements development process", Proceedings of ProSim Workshop '99, Portland, OR, June 1999

[8] F. Stallinger, P. Grünbacher, "System dynamics modelling and simulation of collaborative requirements engineering", *Journal of Systems and Software*, December 2001

Goal-oriented Composition of Software Process Patterns

Jürgen Münch
Fraunhofer Institute for Experimental Software Engineering
Sauerwiesen 6, 67661 Kaiserslautern, Germany
Juergen.Muench@iese.fraunhofer.de

Abstract

The development of high-quality software or software-intensive systems requires custom-tailored process models that fit the organizational and project goals as well as the development contexts. These models are a necessary prerequisite for creating project plans that are expected to fulfill business goals. Although project planners require individual process models custom-tailored to their constraints, software or system developing organizations also require generic processes (i.e., reference processes) that capture project-independent knowledge for similar development contexts. The latter is emphazised by assessment approaches (such as CMMI, SPICE) that require explicit process descriptions in order to reach a certain capability or maturity level. Among other concepts such as polymorphism, templates, or generator-based descriptions, software process patterns are used to describe generic process knowledge. Several approaches for describing the architecture of process patterns have already been published (e.g., [7]). However, there is a lack of descriptions on how to compose process patterns for a specific decelopment context in order to gain a custom-tailored process model for a project. This paper focuses on the composition of process patterns in a goal-oriented way. First, the paper describes which information a process pattern should contain so that it can be used for systematic composition. Second, a composition method is sketched. Afterwards, the results of a proof-of-concept evaluation of the method are described. Finally, the paper is summarized and open research questions are sketched.

1. Introduction

Providing generic processes for organizations and deriving custom-tailored software development processes for projects is a challenging task: Typically, contexts and technology are changing often, new state-of-the art knowledge should be considered, conformance to standards often needs to be guaranteed, experience from past projects should be captured, and distributed development (such as offshoring) has a growing impact on software development processes. Besides these aspects, process models need to be designed, described, evolved, and introduced in a way that they are really used in practice. This is a challanging task requiring methodological, technological, and organizational support for software developing organizations.

Two levels of abstraction can be distinguished: On the one hand, generic processes capture project-independent process knowledge that is relevant for certain application domains (e.g., space software) and/or development contexts (e.g., large projects). On the other hand, project-specific process models describe activities for concrete projects. Typically, project-specific models are derived from generic models through tailoring.

Both generic models and project-specific models need to be evolved: Process-specific models are often evolved through refinement or replanning activities during the course of the project. Generic processes can be evolved, for instance, by including feedback from the execution of projects that are in the scope of the generic model.

The tailoring of software process models to project constraints requires an understanding of process variations and knowledge about when to use which variation. Typically, the following logical steps are needed before the customization is performed [11]:

- Possible process alternatives need to be elicited and explicitly described.
- Process alternatives need to be characterized and constraints/rules on their use need to be formulated. This requires a deeper understanding of the appropriateness of the process alternatives for different contexts and their effects in these contexts.
- Before the start of the project, a characterization of the project context and its goal is necessary, describing the information needed to select process alternatives.

The approach described in this article is based on an engineering-style development paradigm. This means that planning is based on explicit experience from past

projects (i.e., models) and is done in a goal-oriented way. The performance of development activities should adhere to the plan and appropriate means should be used to control and avoid plan deviations or to do replanning in a systematic way. Finally, experience gained during project execution should be analyzed and captured for future projects. Due to the context-orientation of software development (i.e., development activities are unique for a specific context), models need to be customizable to contexts and their scope of validity should be defined. This means that the applicable contexts and the degree of evaluation must be attached to a model.

Software process patterns can be seen as process model fragments that capture relevant process information for solving typical software or system engineering problems in specific contexts. Patterns are a reuse mechanism for software knowledge. The origins of patterns lie in the building architecture domain [1]. In software engineering, first patterns were used in the area of software design [5]. Based on typical characteristics of these domains, patterns seem to be appropriate for capturing knowledge, at least knowledge about creative activities. This could be seen as a hint that patterns are also an appropriate means for capturing knowledge about software development processes that typically also include highly creative activities. Besides process patterns, patterns are nowadays also used in other software engineering areas such as requirements engineering (e.g., requirements engineering patterns [8]) or organizational structuring (e.g., organizational patterns [3][9]).

Applying software process patterns in the context of project planning requires a set of patterns, a schema for characterizing those patterns, an approach for selecting appropriate patterns, an approach for composing the selected patterns, and, finally, a technique for integrating those patterns on the level of a process modeling notation. This paper focuses on the composition. Approaches for characterization and selection are described, for instance, in [2],[6],[12]. An approach for integrating patterns (or process fragments) on the level of a formal process modeling language is described in [11].

2. Process Patterns and Goals

We define a process pattern as a reusable fragment of a process model or a combination of process models that represents an activity or a set of activities, and that is described together with the following information (see Figure 1):

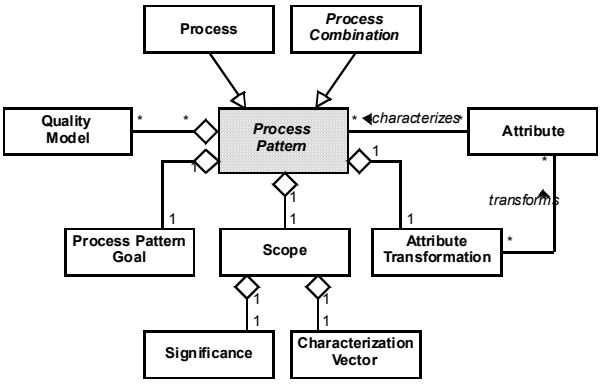


Figure 1. Process Pattern Architecture.

- Characterization vector: a set of characterizing attributes C and an assignment of values C(t) at a certain point in time t.
- Significance: information about the degree of validation of the pattern (e.g., information about the number of usages of the pattern in the context defined by the characterization vector).
- Process pattern goal: an atomic goal describes a restriction on the attributes from the characterization vector by using a < b, a <= b, a > b, a >= b, a = b or a ∈ M. A complex goal is a logic combination of atomic goals or complex goals using the operators "&" (and), "|" (or), "¬" (negation), "⇒" (implication) and "⇔" (equivalence). Process pattern goals can be used to describe the goal that can be reached by a pattern in a certain context. If the pattern is composed of other patterns, the goal describes the goal that can be reached by the composition. An example goal is ((maximal defect rate < 0.8 %) & (service level = high) & (test-effort = f(design-complexity)))
- Attribute transformation: An attribute transformation describes a change of attributes that is caused by the application of a process pattern. An example is reliability:=$_p$ reliability * 1.15, where p is the process pattern.
- Quality models: models describing cause-effect relations (e.g., a prediction model for test effort based on design complexity) can be associated to a pattern.
- Process combination: Process patterns can be combined from other process patterns using composition operators (see below).

The description of pattern goals, attribute transformations and quality models should ideally be based on empirical evidence. Also, process simulation could be used to get relevant knowledge about the effects of process patterns in certain contexts.

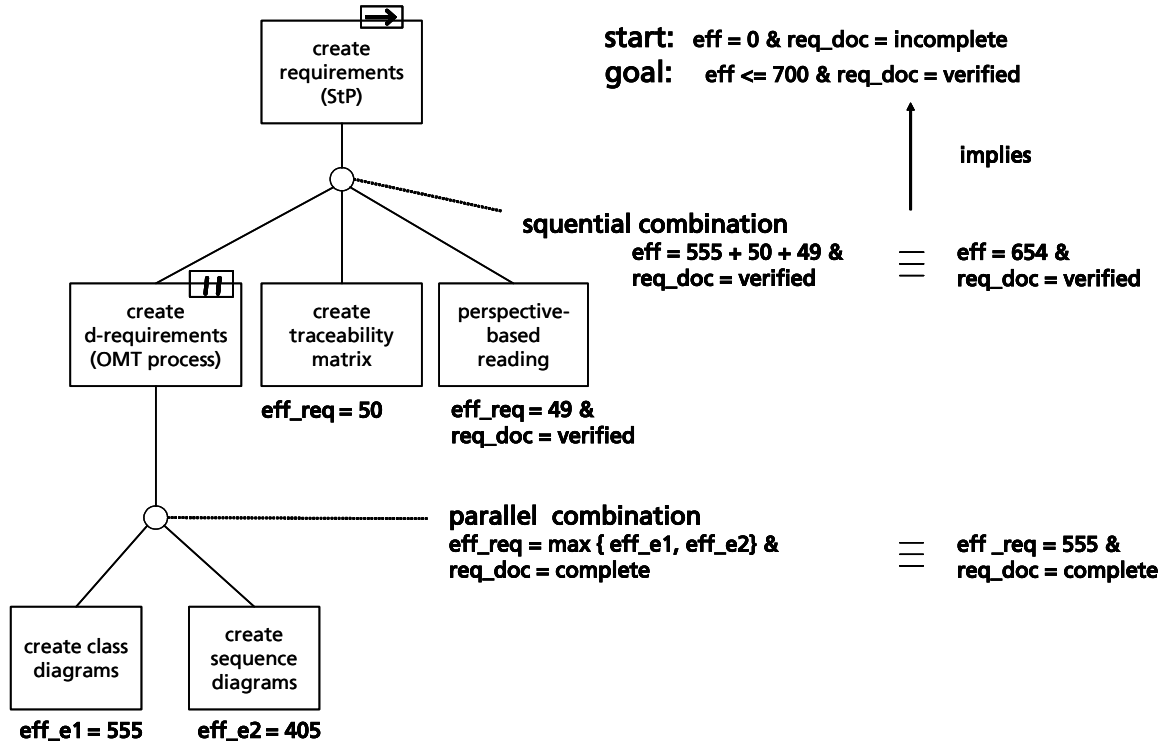


Figure 2. Goal-oriented Composition of Process Patterns.

3. Process Pattern Composition

The starting point for a goal-oriented process pattern composition are project goals that can be formulized analogous to the process pattern goals. The attributes that can be used in the goal definition are a subset of the characterizing attributes of the process patterns and the attributes describing the project context.

For the composition of process patterns, the following composition operators have been defined: sequential combination, parallel combination, conditional combination, and iterative combination. These operators describe how process patterns can be composed and the implications of the compositions on the attribute mappings. The iterative combination can be defined recursively by using the conditional combination:

while (c) t; ≡ if (c) t; while (c) t;

The overall composition method consists of the following steps:

1. Composition of process patterns by replacement, refinement, and augmentation. Based on an initial process pattern, this step creates a combination of process patterns. How process patterns can be combined is constrained. Such constraints are, for instance, that two process patterns can only be combined if the same tool is used or that a logical control flow sequence for a set of process patterns is necessary because of necessary product flows. Such constraints are described via so-called composition networks that need to be established for application domains (see [10] for further details).
2. Verification of the process pattern combination by using formal semantics. During this step, the question is answered of whether the current process pattern combination fulfills the project goal (or sub-goals).
3. If the verification is successful, the formal process descriptions of the process patterns need to be integrated. The approach described in [11] and [13], which also integrates the product flows between the process patterns, could be used. This step includes the transformation of processes from the pattern level to the process instance level, i.e., the customization of the integrated process patterns to the project environment. If the verification is not successful, a new composition needs to be chosen (i.e., step 1 starts again).

Verification and goal-oriented composition of process patterns can now be defined in the following way: Let g be a project goal and p a process pattern combination. Let [p] denote the attribute transformation that is caused by the process pattern combination p. Let s be the values of all characterizing attributes at the beginning of the project. Then we can define:

1. The verification of a goal is the proof that $[p](s) \Rightarrow g$
2. The goal-oriented composition of process patterns is the creation of a process pattern combination p so that $[p](s) \Rightarrow g$

The details of the method are described in [10]. Figure 2 gives an example of the method: Here, the project goal is the development of a verified requirements document with an overall effort (eff) of less than 700 hours. The starting conditions are that no effort has been consumed and the requirements document is incomplete. Below the patterns that are displayed using rectangles, the attribute transformations are given. Subsequently, a parallel combination and a sequential combination are applied and the effects are calculated. Showing that the expression "effort = 654 hours and requirements_document = verified" implies the goal completes the planning and demonstrates that a process combination has been found that promises to be a good basis for a project plan (based on the empirical evidence captured in the attribute transformations of the process patterns). For simplicity reasons, the attribute transformations in the example are described on the instance level (i.e., effort is given in absolute values). Typically, an additional mapping from the process pattern level to the instance level is necessary. For example, the effort needed for conducting a process pattern could be given as a percentage value of the overall effort, which is estimated during project planning. This effort needed for conducting a pattern may also be determined by a function on a value of the project characterization vector.

4. Evaluation

The method has been applied in a case study that focused on the planning of a large development effort for an embedded automation system. 42 process patterns were defined and applied. The goal of the evaluation was to qualitatively show the applicability of the method and to get feedback on how to improve the method. For the project two alternative development tools (Rapsody and StP) were considered that influenced many development activities. Therefore, two clusters of process pattern combinations were built. In the first cluster, 32 process pattern combinations needed to be assessed with respect to their suitability to fulfill the goal. In the second cluster, 15 alternatives needed to be assessed. In each cluster, less than 5 verification steps were necessary. Several combinations have been proven to be suitable. It was possible to choose among the results depending on priorities (e.g., minimize planned effort, optimize defect density rate). The project was conducted according to the plan using more than 20 students. The monitoring of the project data showed high plan adherence (based on qualitative judgement).

In addition, step 3 of the composition method has been evaluated in several case studies. This is described in detail in [10].

5. Conclusions and Future Research Topics

This paper sketched a method for composing process patterns in a goal-oriented way and showed initial evaluation results. The application of the method has shown that it is possible to provide mechanisms for systematically applying and using process patterns during project planning. Moreover, the method provides mechanisms for verifying that a certain combination of process patterns fulfills a project goal in a specific context. It should be mentioned that as with all models, the quality of the results heavily depends on the validity (i.e., mainly the empirical basis) of the models. Challenges for future research in this area are, for instance,

- the development of a graphical representation for process pattern combinations (especially mechanisms to visualize variability),
- automated support for verifying process pattern combinations against goals,
- gaining experience about appropriate granularity levels for process descriptions,
- generating process documentations from process pattern combinations,
- integrating patterns for software development activities and patterns/processes for system development activities (e.g., development activities in mechanical engineering).

Acknowledgement

The author would like to thank Jens Heidrich from the Fraunhofer Institute for Experimental Software Engineering (IESE) who contributed to the implementation of the method, and Sonnhild Namingha from the Fraunhofer Institute for Experimental Software Engineering (IESE) for reviewing the first version of the article.

References

[1] Linda C. Alexander and Alan M. Davis. Criteria for Selecting Software Process Models. In Proceedings of the 15th Annual International Computer Software & Applications Conference, pp. 521-528. IEEE Computer Society Press, Tokyo, 1991.

[2] Ralf Bergmann, Hector Muñoz-Avila, Manuela Veloso and E. Melis. Case-based Reasoning applied to Planning. In M. Lenz, B. Bartsch-Spörl, H.D. Burkhard

and S. Wess, Eds., Case-Based Reasoning Technology: From Foundations to Applications. Lecture Notes in Artificial Intelligence 1400, Springer Verlag, 1998.

[3] James O. Coplien. A Development Process Generative Pattern Language, chapter 13, pages 183--237. Addison-Wesley, Reading, MA, 1995.

[4] Khaled El Emam, Nazim H. Mahavji and K. Toubache. Empirically Driven Improvement of Generic Process Models. In Proceedings of the 8th International Software Process Workshop, 1993.

[5] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Design Patterns – Elements of Reusable Object-Oriented Software. Addison Wesley, 1995.

[6] Soeli T. Fiorini, Julio Cesar Sampaio do Prado Leite and Carlos José Pereira de Lucena. Reusing Process Patterns. 2nd International Workshop on Learning Software Organizations (LSO), 2000.

[7] Mariele Hagen and Volker Gruhn, Process Patterns – a Means to Describe Processes in a Flexible Way, Proceedings of the 5th International Workshop on Software Process Simulation and Modeling (ProSim 2004), Edinburgh, Scotland, United Kingdom, pp.50-56, May 24-25, 2004.

[8] Kathrin Lappe et al., Requirements Engineering Patterns – An Approach to Capturing and Exchanging RE Experience. Final Report from the WGREP, DESY 2004, Hamburg, Germany.

[9] M. Lipshutz, R. Creps und M. Simos. Organizational domain modeling (ODM) tutorial, January 1997.

[10] Jürgen Münch. Muster-Basierte Erstellung von Software-Projektplänen (in German). PhD Theses in Experimental Software Engineering, Vol. 10, ISBN: 3-8167-6207-7, Fraunhofer IRB Verlag, 2002.

[11] Jürgen Münch. Transformation-based Creation of Custom-tailored Software Process Models. Proceedings of the 5th International Workshop on Software Process Simulation and Modeling (ProSim 2004), Edinburgh, Scotland, United Kingdom, pp.50-56, May 24-25, 2004.

[12] Rubén Prieto-Díaz und Peter Freeman. Classifying Software for Reusability. IEEE Software, 4(1): 6-16, IEEE Computer Society, January 1987.

[13] Markus Schmitz, Jürgen Münch und Martin Verlage. Tailoring of large process models on the basis of MVP-L (in German). In Günther Müller-Luschnat, Sergio Montenegro, Ralf Kneuper, eds., Proceedings of the 4th workshops of the section 5.1.1 (GI), Berlin, March 1997

Towards an Interactive Simulator for Software Process Management under Uncertainty

Thomas Birkhoelzer, Christoph Dickmann, Juergen Vaupel, Joerg Stubenrauch

***Abstract*—The management of a software producing organization can be considered as controlling a complex system. However, the dynamics and the parameters of this system are often unknown, or just vaguely known in the best case. Nevertheless, managers need to operate and decide in their daily practice using "uncertain" or "not validated" information.**

Therefore, a simulator is developed in this work, which includes three essential building blocks to simulate an environment of such uncertainties: 1. Parameters with random deviations; 2. Time-varying parameters causing structural different behavior; 3. Dynamics such that short-term effects are opposite to long-term effects.

The purpose of such a simulator is to raise the awareness and understanding of decisions and their associated effects, like the investment in a specific process area and the effect on the software development organization.

***Index Terms*—Capability maturity model, software process simulation, uncertain process information.**

I. Introduction

A. Intention

FROM a system theoretic point of view, the management of a software producing organization can be described as the task of controlling the system of a "software producing organization". Such a system is governed by a set of internal states (capabilities) and produces a set of outputs (business measures). Management control means to balance these outputs to achieve the business goals.

Process simulation and process simulators are used as tools to gain understanding and knowledge of such systems by tests and experiments in a virtual form [4], e.g. to evaluate strategies [3], [5], [14], to compare process alternatives [6], [12], [15], or for teaching and training [7], [10], [11], [13].

Software development management decisions can be more solid if based on a system of quantitative measures. In reality, a mix of qualitative and quantitative business indicators can be assumed to serve as a decision basis. However, the quantitative analysis of software processes, which is necessary in the context of such simulations, often reveals that there is only very limited real-world quantitative information available to build or verify the underlying models. Often, a lot of the parameters in the simulation models need to be estimated or derived indirectly. Validation of the parameters or the models in general is often mentioned as one of the difficult or missing tasks [8], [12]. Most often, this is not due to methodological difficulties but due to missing data.

Yet, this uncertainty is not a flaw of the simulation approach – although sometimes discovered in this context – but a characteristic of software development management in general: Software managers make decisions about processes even without precise knowledge about the quantitative mechanisms and effects. Comprehensive quantitative process information is part of the maturity level four in CMM and CMMI (Capability Maturity Model Integration), and only about 20% of the reporting organizations have reached that level [16]. This means that 80% of organizations operate without such information being used in a comprehensive, planned and sophisticated manner!

Moreover, due to the variability of software tasks and constantly changing development conditions, it seems almost inevitable to operate under a certain level of uncertainty: Even the most precise data gathered for one project or time period might not be valid for the next one. Furthermore, it normally needs a longer learning period to build up the expertise that allows for a qualification of the quantitative project measures for a specific organization. This process of calibration for the quantitative project measures is specific for each individual organization and can therefore not be shared between different organizations.

Therefore, this work presents a simulation approach that is designed to mimic such an uncertain and changing environment. Such a simulator is intended to raise the user's awareness for these effects, the underlying mechanisms and their results. It can be used for training or teaching of current or future stakeholders in software production, and help to build up confidence in decisions to be taken, e.g. by simulating best case and worst case scenarios.

B. Background

In [11], an interactive simulator was presented, which allows a user to act in the role of a manager of a software

Manuscript received February 3, 2005.

T. Birkhoelzer is with the Department of Electrical Engineering and Information Technology, University of Applied Science, Konstanz, Germany (phone: +49 7531 206239; e-mail: birkhoelzer@fh-konstanz.de).

C. Dickmann is with Siemens Medical Solutions, Erlangen, Germany (e-mail: christoph.dickmann@siemens.com).

J. Vaupel is with Siemens Medical Solutions, Erlangen, Germany (e-mail: juergen.vaupel@siemens.com).

J. Stubenrauch is with the Department of Electrical Engineering and Information Technology, University of Applied Science, Konstanz, Germany (e-mail: joerg.stubenrauch@gmx.de).

producing organization. The manager's task is to best meet strategic business goals by iteratively investing in and improving software development processes. The simulation model's internal states represent capability levels of CMMI key process areas [9]. The inputs in the model are abstract "investments in key process areas" covering financial as well as non-financial efforts. One investment point as abstract fiscal unit is the effort necessary to raise the capability level of the associated key process area by one level. The model's outputs are balanced scorecard [2] business measures that provide the basis for the next process improvement investment decision.

The resulting model is a non-linear, time-discrete, state-space system with $\bar{x}_t = (x_{1,t}, \ldots, x_{n,t})$ denoting the *n*-tuple of the internal state variables, $\bar{u}_t = (u_{1,t}, \ldots, u_{l,t})$ the *l*-tuple of input variables, and $\bar{y}_t = (y_{1,t}, \ldots, y_{m,t})$ the *m*-tuple of output variables:

$$x_{i,t+1} = \mathrm{norm}\left(\lambda_i \cdot x_{i,t} + (\lambda_i - 1) \cdot s_{i,t}\right) \quad (1)$$

$$s_{i,t} = u_{i,t} + \sum_j \beta_{ij} \cdot x_{j,t} + \sum_j \gamma_{ij} \cdot \mathrm{gate}(x_{l_{ij},t}, \nu_{ij}, \mu_{ij}) \cdot x_{j,t} \quad (2)$$

$$y_{i,t} = \sum_j \alpha_{ij} \cdot x_{j,t} + \sum_j \chi_{ij} \cdot \mathrm{gate}(x_{l_{ij},t}, \sigma_{ij}, \rho_{ij}) \cdot x_{j,t} \quad (3)$$

with

$$\mathrm{norm}(arg) = \begin{cases} 5 & \text{for} \quad arg > 5 \\ arg & \text{for} \quad 0 \le arg \le 5 \\ 0 & \text{for} \quad arg < 0 \end{cases} \quad (4)$$

$$\mathrm{gate}(arg, \nu, \mu) = \frac{1}{1 + e^{-\nu(arg - \mu)}} \quad (5)$$

Equations (1) and (2) are separated to make the underlying structure transparent: the term $s_{i,t}$, consisting of the investment input and the feedback from other states is the driving force of the first order dynamics of (1). The parameter λ_i, with $0 < \lambda_i < 1$, models the time constant of the dynamics of the *i*-th process area, representing the "ease and speed" of process improvements in this process area. For values of λ_i close to zero, the state reacts fast on changes of $s_{i,t}$. The closer the value of λ_i is to 1, the slower is the reaction. A parameter value $\lambda_i = 0$ would mean "no delay" or "no dynamic" respectively. For $|\lambda_i| > 1$ the system would be unstable, in the case $-1 < \lambda_i < 0$ the state would oscillate.

In CMMI, the capability level of each process area is characterized by a value from zero to five:, zero is the smallest possible value (non-existing or incomplete respectively), five the maximal one (optimizing or perfect). Therefore, the state values $x_{i,t}$ are restricted as well by the range from zero to five by the function $\mathrm{norm}(\cdot)$ defined by (4).

In (2) the parameters β_{ij} and γ_{ij} serve as weights for the influence of the *j*-th process area on the *i*-th process area. A linear influence is expressed by the parameter β_{ij}. The term $\gamma_{ij} \cdot \mathrm{gate}(x_{l_{ij},t}, \nu_{ij}, \mu_{ij})$ is used to model that the influence of some process areas depends of a certain capability level of another process area as a prerequisite. In CMMI, this is the background of the assignment of process areas to maturity levels within the staged representation: the process areas of the lower maturity levels are prerequisites of the process areas of the higher maturity level. The gating function $\mathrm{gate}(arg, \nu, \mu)$ defined by (5) translates this concept of a prerequisite in a continuous mathematical form: for small values of arg, the function value is close to zero, for large values of arg, the function is close to one, for $arg = \mu$ the function value is 0.5. Therefore, the parameters μ_{ij} express the "switching values", the parameters ν_{ij} the slope of the transitions.

Equation (3) has the same structure as (2) where α_{ij} denotes the strength of a linear influence and χ_{ij} of a non-linear, gated one, of the *j*-th process area on the *i*-th business measure. For modeling simplicity, all business measures are additionally normalized to the range zero to one and such that the value zero denotes the worst and the value one the best case (e.g. business measure Malfunction = 1 represents "no malfunctions at all").

A difference equation as described by (1) needs initial conditions. These describe the status of the organization at simulation start. By specifying appropriate initial conditions, different organizational starting points can be used for the interactive simulation task, e.g. a rather mature organization or a start-up organization with no capabilities at all.

In this form, the mathematical model is a strong abstraction and simplification of real-world mechanisms. On purpose, it reflects essential properties of the dynamics of a software producing organization only, and does not model "real world" mechanisms directly.

The model is implemented as an interactive simulator, i.e. at each time step, the user can allocate the available budget (which is derived based on the actual business measures) to investments in key process areas during the next time period.

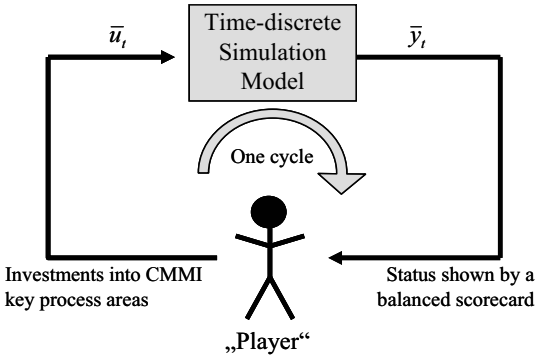


Fig. 1. Interactive Simulator

II. BUILDING BLOCKS TO MIMIC UNCERTAINTY

A system as described in Section I.B and [11], with simple first order time dependencies and few time invariant internal feedbacks, i.e. constant dependencies between internal states, is relatively "easy" to control, i.e. to manage. The direction of effects (positive or negative) of decisions and strategies are immediately visible in the outputs and do not change during the simulation. Therefore, strategies can easily be probed, evaluated and eventually corrected in few trials or time steps.

However, such a model does not include fundamental real-world behavior and thus may simplify more than is desired even for theoretical or didactic reasons. There are at least three effects, which add substantial realistic variability on model effects.

A. Random parameter deviations

The model parameters are used to adapt the model to a specific organization's conditions. If sufficient data would exist, this would be a standard parameter identification problem. However, different organizations have different parameter sets.

For a manager, who joins a new organization, chances are good that this organization differs more or less from her or his previous experiences. Thus, the parameters of the new organization are different from his previous experiences and difficult to estimate.

Such organizational uncertainties of software management are mimicked in the model by random parameter deviations. This is a known, standard modeling technique. The range of deviation can be specified for each parameter. Each time a new simulation round is started, a new parameter set is chosen. This forces the user to watch closely the evolvement of the model (or the organization respectively) and not to just blindly rely on past recipes.

B. Time-varying parameters

Unlike an assembly line, a software producing organization is not a time invariant system. The parameters of the software production vary considerably between projects, each new technology or methodology introduced will change some or all of the process parameters. Such changes over time are assumed to be monotonous for a certain time period: For instance, introducing a more effective teamwork or workflow environment will improve the support between processes. The gradual and monotonous character of this change is due to the gradual adoption process.

Thus, software management has to be aware that, even within the same organization, data or experiences collected in the past might not be valid in the future.

Therefore, the second building block to mimic the uncertainties of software management is the introduction of time-varying parameters.

Mathematically, this is modeled by a linear transition between two sets of parameters:

$$\begin{aligned} \beta_{ij}^{(actual)} &= f_t \cdot \beta_{ij}^{(Set1)} + (1 - f_t) \cdot \beta_{ij}^{(Set2)} \\ f_t &= f_{t-1} + q_t \cdot \Delta f \end{aligned} \tag{6}$$

Moreover, f_t is limited to the interval $[0,1]$. The parameter sets can be specified in the model file, the transition speed Δf modeling the speed of change can be configured in the simulator. The transition can be turned on or off at any point during the course of the simulation by the switch $q_t \in \{0,1\}$.

C. Uncertain short–term dynamic effects

In (1), the model state dynamics are described by a first order difference equation. The step response (i.e. the time response after a step increase in the input) of such a first order difference equation with $s_{i,t}$ regarded as input is shown in Fig. 2. (Note that this is a simplification, since $s_{i,t}$ is usually not constant due to state dependencies.)

Process improvements, however, can show a structural different behavior in which the short-term dynamic effects of decisions are opposite to the long-term effects. Consider the example, where an investment, i.e. change, in a key process area "disturbs" the routine in the short-term causing the maturity to decline first. Only after a while, the expected and intended positive effect (i.e. investment raises the capability level) becomes visible. The most well-known example of such behavior in Software Engineering is Brooks' Law: "Adding manpower to a late project makes it later" [1]. Due to similar reasons as mentioned in [1] (change consumes resources), it is expected that most of the complex processes or activities show such a behavior.

As an additional effect, processes can slightly deteriorate in the long-term compared to their maximal performance. This might be accredited to the build-up of inflexibility and bureaucracy.

Fig. 3 shows the step response of a system with these properties, i.e. an initial undershoot followed by a slight overshoot.

Such dynamics can be modeled by a third order difference equation, i.e. equation (1) is replaced by the following dynamics:

$$\begin{aligned} x_{i,t+1} = \mathrm{norm}(&\lambda_{i,0} \cdot x_{i,t} + \lambda_{i,1} \cdot x_{i,t-1} + \lambda_{i,2} \cdot x_{i,t-2} \\ &+ \kappa_{i,0} \cdot s_{i,t} + \kappa_{i,1} \cdot s_{i,t-1} + \kappa_{i,2} \cdot s_{i,t-2}) \end{aligned} \tag{7}$$

In case of such a dynamic, it is necessary to stick to a strategy (an investment) even if it looks like a wrong strategy during the initial steps due to the initial decrease (the undershoot). However, once understood, this would be as easy to cope with as the simple first order behavior: just assume the opposite short-time effect for assessing a strategy, i.e. a strategy would only be good, if it results in a short term decrease.

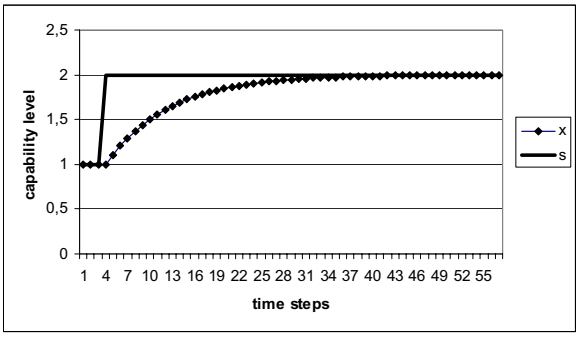


Fig.2. Step response of a first order difference equation, $\lambda_i = 0.89$.

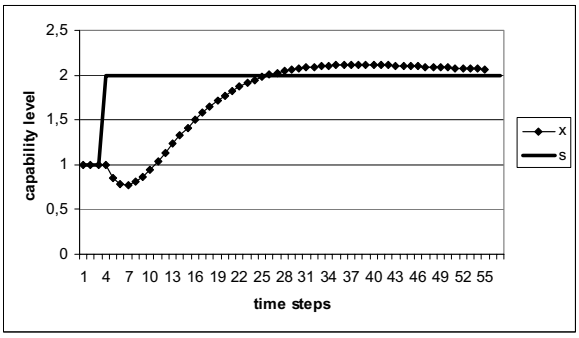


Fig. 3. Step response of a third order difference equation,
$\lambda_{i0} = 2.63, \lambda_{i1} = -2.3\, \lambda_{i2} = 0.669,$
$\kappa_{i0} = -0.15, \kappa_{i1} = 0.327, \kappa_{i2} = -0.176$.

Real world settings are more complicated due to a mix of these two types of behavior. Some process area have a step response as shown in Fig. 2, others as shown in Fig. 3. Then it is more difficult to find adequate mitigation strategies because it may not be clear whether a strategy is right despite initial negative effects, or whether it is wrong for the whole decision period.

Therefore, the third building block to mimic the uncertainties of software management is the extension of the dynamical model such that some process areas have a step response similar to Fig. 2 and others similar to Fig. 3.

III. Exemplary Results

Below, the effects of time-varying parameters and higher order dynamics as discussed in Section II.B and II.C are demonstrated in two exemplary scenarios. The random parameter deviations introduced in Section II.A primarily serve to vary the task between simulation runs, not within one simulation of the same scenario. Therefore, they are omitted here.

For both scenarios, the structure of the simulation model presented in [11] is used. It consists of 15 process areas deduced from [9] and 27 business measures arranged in six perspectives of a balanced scorecard [2]. Only the summary scores of the six perspectives are shown here in order to avoid a detail overload.

Three process areas are modeled by a third order difference equation with parameters according to Fig. 3: Requirements Development, Technical Solution, and Organizational Process Focus. These process areas highly depend on specific knowledge and highly skilled persons. Therefore, improvements require substantial resources from within the process causing the initial decline in the step response. All other process areas are modeled by first order difference equations.

A. Investment in Requirements Development

The first scenario simulates an organization that has about reached maturity level three. However, Requirements Development has been neglected so far, i.e. there were no direct investments in this process area yet. Note that the process area has nevertheless reached a capability level of two due to the support of other process areas.

As a next step in the build-up of the organization, the customer orientation should be improved. Therefore, investments in Requirements Development are set to three. The result of this strategy after 10 time steps is shown in Fig. 4. The perspectives Innovation (left, lower graph) and Customers (middle, lower graph) show a characteristic ditch, which is due to the evolution of Requirements Development as shown in Fig. 5: first a drop in performance and then a gradual improvement. This means, during the first time steps (values in the middle of the chart graphs), the investment in Requirements Development seemed to be a wrong strategy, since the respective measures slumped initially (see Fig. 4). Only by sticking to the strategy, the investments finally pay off.

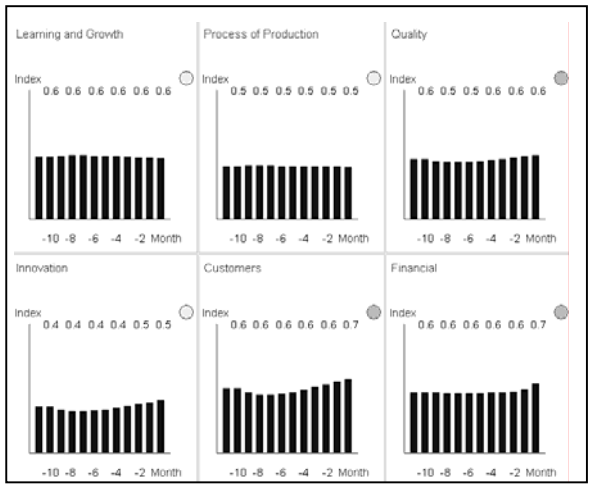


Fig. 4. Step response of a balanced scorecard summary view.

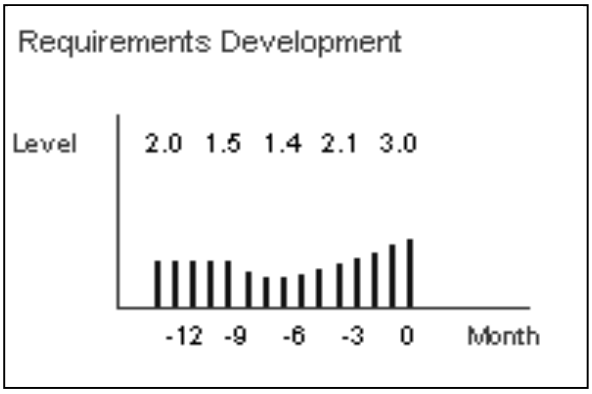


Fig. 5. Step response of process area Requirements Development

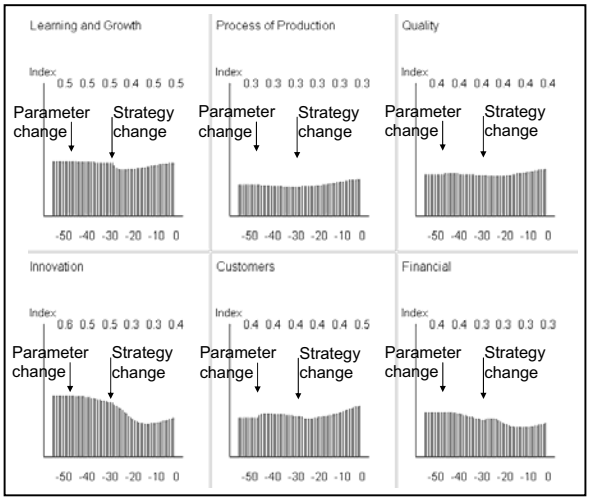


Fig. 6. Business measures of an evolving company.

B. *From teams to organizations*

The second scenario considers an organization that is a small and highly innovative company. In such an environment, Integrated Teaming (e.g. agile development and pair programming) is a key success factor for quickly reacting to changes in requirements and technical approaches. On the other hand, improvements in the process area Organizational Process Focus will cause less improvement in other process areas in such a small organization than in a larger organization.

Therefore, a management strategy with high investments in Integrated Teaming (five investment points) and low investments in Organizational Process Focus (one investment point) is successful and stable, which is reflected in the simulation by stable states and a stable budget.

Over time, such a business evolves towards a bigger organization, mature markets, and legacy products. For such an organization the relative weight (significance) of Integrated Teaming decreases and the weight of Organizational Process Focus increases. This is represented in the simulation by appropriate changes of the parameters in the model.

Typically, this is a gradual change over many time steps, i.e. a typically application of the mechanisms described in Section II.B. To exaggerate this effect for the presentation, the parameters are changed in one time step only, i.e. $\Delta f = 1$ in (6).

After this change, the management strategy, which was previously successful, is not sustainable anymore: It results in a small, constant decline, see Fig. 6. Without countermeasures, the budget will be consumed and the organization may go bankrupt.

To respond to these variations, the management changes its strategy after 20 time steps: the investments in Organizational Process Focus are increased to three and the investments in Integrated Teaming decreased to three. With this new strategy, another 30 time steps are simulated. Fig. 6 shows the complete course after 50 time steps. The organization finally recovers, but starts at a lower business level again. This is due to the fact that the build-up of Organizational Process Focus has considerable time delay even exhibiting an initial drop in the capability level, see above.

IV. Conclusion

In this work, extensions of a model for simulating overall software development in a whole organization are discussed, which are intended to address some of the real world complexity of software management caused by the inherent uncertainty of the effects of decisions. These extensions were designed to mimic the following management problems:

- Random-organizational deviations make it difficult to rely on management "recipes" that were valid in the past.
- Time-varying development conditions make it challenging to extrapolate from past results for estimating future effects, even within the same organization.
- Different short-term adaptations and behaviors (caused by higher order dynamics) make it challenging to probe and evaluate strategies on the basis of their current or short-term effects only.

The intention of the simulator is to simulate the potential effects in order to help understanding potential mitigation conditions, not to propose a solution or answer to these challenges but.

The results presented in Section IV indicate that the simulator can be used as a tool to raise and sharpen the awareness of the complexity of dependencies and variations over time in software development organizations.

References

[1] F. Brooks, *The Mythical Man Month*, Addison-Wesley, 1995.
[2] R. S. Kaplan and D. P. Norton, *The Balanced Scorecard*, President and Fellows of Harvard College, 1996.
[3] C. Lin, T. Abdel-Hamid, and J. Sherif, "Software-Engineering Process Simulation Model (SEPS)", *Journal of Systems and Software 38*, Elsevier, New York, 1997, pp. 263-277.
[4] M. L. Kellner, R. J. Madachy, and D. M. Raffo, "Software Process Modeling and Simulation: Why? What? How?", *Journal of Systems and Software 46*, Elsevier, New York, 1999, pp. 91-105.
[5] J. Williford and A. Chang, "Modeling the FedEx IT Division: A System Dynamics Approach to Strategic IT Planning", *Journal of Systems and Software 46*, Elsevier, New York, 1999, pp. 203-211.

[6] D. M. Raffo and M. I. Kellner, “Modeling Software Processes Quantitatively and Evaluating the Performance of Process Alternatives”, in *Elements of Software Process Assessment and Improvement*, editors K. E. Emam and N. Madhavji, IEEE Computer Society Press, Los Alamitos, California, 1999, pp. 297 -341.

[7] A. Drappa and J. Ludewig, “Simulation in Software Engineering Training”, *Proceedings of the 22nd International Conference on Software Engineering*, ICSE, Limerick, Ireland, June 2000, pp. 199-208.

[8] D. M. Raffo and M. I. Kellner, “Empirical Analysis in Software Process Simulation Modeling”, *Journal of Systems and Software 53*, Elsevier, New York, 2000, pp. 31-41.

[9] CMMI Product Team, “Capability Maturity Model Integration (CMMI-SE/SW/IPPD, V1.1), Continuous Representation”, Software Engineering Institute, Pittsburgh, 2001.

[10] D. Pfahl, M. Klemm, and G. Ruhe, “A CBT Module with Integrated Simulation Component for Software Project Management Education and Training*”, Journal of Systems and Software 59*, Elsevier, New York, 2001, pp. 283-298.

[11] Th. Birkhölzer, L. Dantas, C. Dickmann, J. Vaupel, „Interactive Simulation of Software Producing Organization's Operations based on Concepts of CMMI and Balanced Scorecards“, *Proceedings of the 5th International Workshop on Software Process Simulation and Modeling* (ProSim 2004), Edinburgh, May 2004, pp. 123-132.

[12] D. Pfahl, M. Stupperich, and T. Krivobokova, “PL-SIM: A Generic Simulation Model for Studying Strategic SPI in the Automotive Industry”, *Proceedings of the 5th International Workshop on Software Process Simulation and Modeling* (ProSim 2004), Edinburgh, May 2004, pp. 149-158.

[13] E. Oh Navarro and A. van der Hoek, “Software Process Modeling for an Interactive, Graphical, Educational Software Engineering Simulation Game”, *Proceedings of the 5th International Workshop on Software Process Simulation and Modeling* (ProSim 2004), Edinburgh, May 2004, pp. 171-176.

[14] F. Padberg, “A Comprehensive Simulation Study on Optimal Scheduling for Software Projects” , *Proceedings of the 5th International Workshop on Software Process Simulation and Modeling* (ProSim 2004), Edinburgh, May 2004, pp. 177-185.

[15] D. Raffo, U. Nayak, S. Setamanit, P. Sullivan, W. Wakeland, “Using Software Process Simulation to Assess the Impact of IV&V Activities”, *Proceedings of the 5th International Workshop on Software Process Simulation and Modeling* (ProSim 2004), Edinburgh, May 2004, pp.197-205.

[16] CMM Appraisal Program, “Process Maturity Profile; Software CMM 2004 Mid-Year Update”, Available: http://www.sei.cmu.edu/sema/profile_SW-CMM.html, 8/2004.

Effective Resource Allocation for Process Simulation: A Position Paper

Mohammad S. Raunak and Leon J. Osterweil
University of Massachusetts at Amherst
{raunak, ljo}@cs.umass.edu

Abstract: **We often simulate processes to be able to reason about, forecast, and plan the best utilization of available resources. As process programmers, we define resources to be the agents that carry out tasks, and the tools and other entities required by agents in order for them to be able to complete their assigned work. Specifying these resources rigorously and allocating them efficiently during process simulation or execution is a non trivial problem. In this paper, we present many hard and interesting issues related to resource management and propose some solution approaches. In particular, we talk about an auction based solution approach, which we feel fits well in different types of process simulation.**

Index Terms— **allocation, process, resource, simulation**

I. Introduction

One of the primary reasons to execute or simulate processes (and many other types of software) is to be able to reason about, forecast, and plan, the best utilization of available resources [kellner99]. Managing resources well translates into better process management, which in turn, ensures better quality in the final products or services resulting from a process. From our perspective as process programmers, we define resources to be the agents that carry out tasks, and the tools and other entities required by agents in order for them to be able to complete their assigned work. Specifying these resources rigorously and allocating them efficiently during process simulation or execution is a non trivial problem.

Our previous work in process programming has focused on specifying the coordination of activities, their execution semantics and artifact flows [wise98, cass99, cass00]. All these issues are undeniably crucial for properly capturing a process and supervising its execution. However, we have always maintained that a clear, precise and complete definition of the resources required by the process, and flexible mechanisms for their allocation, are no less important for process analysis and improvement. We argued the importance of a separate resource manager and laid out a modeling approach in earlier works [rodion99, lerner00]. Over the last few years our experience with process modeling, analysis, and execution have grown, drawing from our work with processes in such diverse domains as digital government, e-commerce, and medical services, as well as with software process. These experiences have provided insights into some more challenging issues and have pointed us toward some promising approaches to address the challenges. In this paper, we present many hard and interesting issues related to resource management and propose some solution approaches.

This research was partially supported by the Air Force Research Laboratory/IFTD and the Defense Advanced Research Projects Agency under Contract F30602-97-2-0032, by the U.S. Department of Defense/Army and the Defense Advance Research Projects Agency under Contract DAAH01-00-C-R231, and by the National Science Foundation under Grant No. CCR-0204321. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied of the Defense Advanced Research Projects Agency, the Air Force Research Laboratory/IFTD, the U.S. Dept. of Defense, the U. S. Army, The National Science Foundation, or the U.S. Government

.

II. Resource Management Challenges

In our work we define a process as (largely) hierarchical structure of tasks, called steps. Each step is to be executed by an agent, whose characteristics are specified as part of the process definition, and a set of auxiliary resources, also specified as part of the process definition [cass00, wise98]. In our approach, the execution of such a process definition entails the binding of a specific agent, and specific resources, identified by means of a search through a resource repository, to each process step, as it comes up for execution. An important premise of our work is that resources, and indeed agents, can be either humans or automated devices (either hardware or software). Thus our process definitions can rightly be viewed as specifications of how humans and automated devices are to be coordinated. This view particularly emphasizes the importance of effective identification of the agents and resources to be selected for binding to the various steps. Invariably the resources that are available are in short supply and are the subjects of contention. Thus, potential delays, bottlenecks, and resource starvation can presumably be avoided or reduced by using resource analysis to identify ways in which resource contention can be alleviated (e.g. by causing tasks to execute in parallel or by skillful sharing of resources).

Another way to achieve better process performance is to schedule the resources based on some allocation strategies. Researchers in operating systems, multi agent systems and

operations research have long looked at different allocation algorithms for better utilization of resources in processes. However, the challenges we face in software and a lot of other processes are unique due to the diverse nature of resources we need to deal with here. Resources like processor time, network bandwidth or completely automated agents have strictly predictable behavior that is relatively less complex to schedule and reason about. The simultaneous presence of humans and automated agents, and the potential diversity in the time granularity of the required resources, make it inherently much more difficult to specify them properly and allocate them effectively with a coherent scheduling scheme.

There are three very important and intricately interconnected tasks in the realm of resource management: modeling the resources, identifying and retrieving the resources and finally, scheduling the resources. The modeling task usually takes place before simulation, potentially in parallel with the process definition work. The modeling or specification of resources, to a large extent, is dependent on the domain for which a process is being programmed. In software processes, for example, we have human agents like programmers, designers, testers as well as non-human resources like compilers, development environments, software licenses etc. On the other hand, a medical emergency room process includes resources like nurses, doctors, orderlies, beds, medicines, and a whole variety of equipments and tools. It is quite challenging to provide a common structure and service architecture that would allow a process programmer to model and allocate resources from these diverse domains coherently.

Identifying and retrieving the resources require capabilities to store resource information persistently and to be able to query them with some rich query language. Specifying and storing both resource types and instances adequately is not always straight forward. The resource manager architecture needs to be flexible enough to support processes from diverse domains that deal with a wide ranging resource types.

Scheduling of resources is primarily required during run time. The modeling and scheduling capabilities that we have previously proposed in [rodion99] have proven effective in programming processes with comparatively less complex resource variety and requirements. Our previous approach worked by statically defining the resource model and letting process steps query that static model. However, we have discovered that agents and resources often have dynamic attributes which may change in real time during the execution of the process. For example, in an emergency department process of a hospital, the nurses (one type of agent/resource) are frequently changing locations and any query based on the propinquity of a resource requires the *resource manager* to accommodate dynamically changing resources states.

Software, medical, and other processes require complex specification of composite resources like design teams or nursing pools which include instances of individual agents, each of which might potentially need to be allocated to several different tasks. While allocation of such resources can be done by modeling capacity of these resources and to allow them to be assigned to multiple tasks in parallel, a more accurate model of such partial allocation would require time based allocation of the agents to each task. In this case, an agent will not be bound to a process step indefinitely; it will be acquired by the step for a specific period of time.

Often we come across process programs where resources are sought in some preferential order. For example, there may be a designated agent for a task; however, other agents with some particular skill can carry out the task in absence of the designated agent. The process definition somehow needs to specify the preferred resource in addition to specifying all the attributes properly that would allow selection of substitute resources that can fill in.

Our experiences with processes and resources have shown that there are process definitions that require the execution to block in case a resource is busy. Allowing such blocking calls to the resource server during an execution or simulation of a process gives rise to the whole dimension of potential deadlocks and starvations.

An important requirement of resource management is the ability to reserve resources for planning purposes. This service requires look-ahead capabilities on the process execution paths. As process programmers, we often need to deal with complex processes with abundant exceptions. Such control flow structures make it very difficult to predict future resource requirements based on which proper reservation can be made. A related requirement for a resource manager is to be able to identify the level of resource redundancy required to ensure safe completion of processes in case of a sudden surge in activities.

Our ongoing work with medical processes has strongly demonstrated the need for resource preemption capability in the resource manager. Resources like doctors, nurses or even hospital beds get preempted and assigned to a new instance of a task with higher priority in an emergency department process. We are convinced that such cases of higher priority work preempting resources from a running low priority task is not uncommon in software processes either. From the process simulation or execution perspective, this brings in a whole new level of complexity.

III. Solution Approaches

In [lerner00], we described a resource model based on *resource classes*, *resource instances*, their *attributes* and *relationships* amongst them. Our subsequent experience with process simulation work has shown that not all the elements of that proposed framework are necessarily effective for resource modeling and allocation in simulations. We thus propose a new approach to resource modeling where entities are primarily categorized as agents and non-agent resources. Our contention is that the association between a process step and its required resources is driven by constraints on the agent or agents responsible for carrying out the task.

Resources usually have different types of constraints. Amongst others, there are “requires” relationships between resource entities which prevent assigning of a resource without also assigning one or more other required resource. This constraint can be addressed by having attributes associated with the resources that would allow the resource

manager to query on a resource attribute to create compound resource collections dynamically at run time.

The other important constraint on resources is brought about by dynamically created composite objects. For example, a programmer pair in XP pair programming practice or a couple of nurses assigned to a 'trauma patient'. We propose to utilize ideas similar to "views" used by database researchers to create abstract tables on the fly. The process programmer will define resource objects and will provide definitions of the types of composition allowed for these resource objects. When a process step specifies a query for one of these composite resource objects, a query processor will join multiple objects to create composite resource on the fly and the Resource Manager will retrieve them atomically.

We also propose the incorporation of timing constructs in our process model, Little-JIL [wise98], in order to provide important information regarding the time within which an agent needs to start its work after being assigned to a task, and the maximum time the agent is given to complete the task. In some cases, we also envision the need for specifying the minimum time required for a task for scheduling purposes.

The resource analysis study is often performed with the objective of identifying the amount of redundancy required for resources to successfully execute processes in case of a surge of activity. We are in the process of collecting real life data for a medical emergency department process to investigate the characteristics of process behavior and resource loads under such circumstances. We also plan to explore process simulation with stochastic input model to create such scenarios.

We will explore the allocation of resources using strategies based on known scheduling algorithms from other areas like operating systems for non-agent resources. However, we propose a novel approach for the assignment of agents to tasks. Agents and groups of agents will be presented with specifications of tasks, time requirements and other resource requirements related to them. Assignment of agents to tasks is to be done by means of an auction. The agents will be required to bid for assignment to these tasks, being incentivized to bid as aggressively as possible, consistent with private valuations based upon their need for work, desire for advancement, or fear of layoffs. The final assignment of tasks will be done in response to determination of the highest bidder. Groups of agents in this scenario will be allowed to communicate amongst themselves in a manner that is similar to the way bidders may collude in different types of auctions to reduce their bid. In our case, however, bidder collusion will be aimed at coming up with the highest bid for a task.

We believe this idea of auctioning is going to be applicable in different resource management issues. The preferential ordering of resources that we have discussed earlier can be achieved based on the bids placed by the different agents (and other resources). We also feel that the requirements of supporting dynamic creation of composite resource objects can be aptly addressed with the utilization of combinatorial auctions. There is a broad literature in Operations Research on different auction formats and their effectiveness. We believe, we can utilize those results and experiment with them in our environment to discover their utility.

IV. Related Works

Different software process programming languages like APEL[establier97], MVP-L[rombach93], ALF[canals94], Statemate[harel90] and Process Weaver[fern93] have used resource managers to facilitate process execution. However, the modeling capabilities in such systems are restrictive and do not provide support for scheduling.

There has been considerable work in operating systems focusing on scheduling techniques of primarily hardware resources [goyal96, shenoy98]. As mentioned earlier, the domains of their work including required timing granularity, differ significantly from resources that include both human and automated agents as well as other entities.

V. Concluding remarks

Modeling as well as scheduling of resources to reason about effective process simulation and resource requirements is a hard problem. There is ample literature in other areas of computer and management sciences that look at the resource management issues from different domain perspectives. It is crucial for our community to take a critical look at the issues of resource management, identify the unique challenges faced by the process programmers while simulating or executing processes, determine what approaches from other areas may or may not work, and propose and evaluate new solution approaches. In this paper, we have tried to motivate such resource management research by pointing to some intricate issues related to resources within a process environment. We have also proposed an auction based solution approach, which we feel will work well in different types of process simulation and execution.

Acknowledgment

We would like to thank Barbara Lerner, Rodion Podorozhny, Anoop Ninan and Joel Sieh for their earlier attempts at developing some initial versions of resource management service for process execution support.

References

[canals94] Canals, G. et. al., ALF: A framework for building process-centered software engineering environments. In Software Process Modeling and Technology, pages 153-185Researhc Studies Press, Sommerset, England 1994

[cass99] Cass, A.G., Lerner, B. S., McCall, E. K., Osterweil, L. J., Sutton, Jr., S. M. and Wise, A. Logically central, physically distributed control in a process runtime environment, *Technical Report 99-65, University of Massachusetts*, Department of Computer Science, November 1999

[cass00] Cass, A.G., Lerner, B. S., McCall, E. K., Osterweil, L. J., Sutton, Jr., S. M. and Wise, A. Little-JIL/Juliette: A process definition language and interpreter, In *Proceedings of the 22nd International Conference on Software Engineering*, 754-757, Limerick, Ireland, June 2000

[establier97] Establier, J., Dami, S. and Amiour. A., APEL: A graphical yet executable formalism for process modeling, *In proceedings of Automated Software Engineering*, March 1997.

[fern93] Fernstrom, C. PROCESS WEAVER: Adding process support to UNIX. *In the second International conference on Software processes*, pages 12-26, 1993 `

[goyal96] Goyal, P., Guo, X. and Vin, H. A Hierarchical CPU Scheduler for Multimedia Operating Systems. *Proceedings of the 2nd Symposium. on Operating Systems Design and Implementation*, pages 107-122, Seattle, Washington, 1996.

[harel90] Harel, D. and Lachover, H. et. al. STATEMATE: A working environment for the development of complex reactive systems. *IEEE Transactions on Software Engineering*, vol 16, issue 4, April 1990.

[kellner99] Kellner, M. I., Madachy, R. J., and Raffo, D. M., Software Process Simulation Modeling: Why? What? How? *Journal of Systems and Software*, vol. 46, no. 2/3, 15 April, 1999

[lerner00] Lerner, B. S., Ninan, A. G, Osterweil L. J., Podorozhny R. M. , Modeling and Managing Resource Utilization in Process, Workflow, and Activity Coordination, Technical Report, Department of Computer Science, University of Massachusetts, Amherst, MA 01003, August 2000. *(UM-CS-2000-058)*

[rodion99] Podorozhny, Rodion, Lerner, B. S. and Osterweil L. J., Modeling Resources for Activity Coordination and Scheduling. *3rd Internation conference on coordination models and languages*, Amsterdam 1999

[rombach93] Rombach, H. and Verlage, M. How to assess a software process modeling formalism from a project member's point of view. *In proceedings of the Second International Conference on Software Processes,* pages 147-159, 1993.

[shenoy98] Shenoy, P and Vin.H, CELLO: A Disk Scheduling Framework for Next-Generation Operating Systems. *Proceedings of the ACM SIGMETRICS*, 1998.

[wise98] Wise, Alexander, *Little-JIL 1.0 Language Report,* Department of Computer Science, University of Massachusetts, Amherst, MA 01003, April 1998. *(UM-CS-1998-024)*

Understanding Open Source and Agile Evolution through Qualitative Reasoning

Juan C. Fernández-Ramil, Andrea Capiluppi, Neil Smith
Computing Department
The Open University
Walton Hall, Milton Keynes MK7 6AA, U.K.
(j.f.ramil, a.capiluppi, n.smith)@open.ac.uk

1. Introduction

The phenomenon of software evolution has been described in the literature [e.g., Lehman 1974; Lehman and Belady 1985] and several models of different nature [e.g., Aoyama 2002; Capiluppi 2003; Lehman et al 2002; Rajlich and Bennett 2000; FEAST] have been proposed to understand and explain the empirical observations. Some of these models purport to be universally applicable to all software development processes. However, the models in the literature were built mainly observing software developed in what has been the traditional centrally-managed Waterfall development process or one its variants. Since these theories were developed, software development methods have themselves evolved and now much software is developed and evolved by using agile methods (such as XP [Beck 1999]) and in open-source environments. It remains a question to be investigated what are the characteristics of software evolution under these new paradigms and whether the existing models form an accurate description of the evolution of software under these new regimes or whether new models will be required. In previous work [e.g., Ramil & Smith 2002; Smith & Ramil 2002, 2003; Smith et al 2004, 2005] the authors have illustrated how qualitative reasoning methods, such a qualitative simulation, provide a useful way to examine the general behaviour of models of software evolution. This extended abstract describes our plans for further research into the topic.

2. Qualitative Reasoning

The vast majority of the research and development in the process simulation field has focused in the use of quantitative techniques such as system dynamics and discrete-event simulation, as exemplified by the majority of the contributions to the ProSim series of previous workshops. In our work we have been pursuing a radically different approach to the empirical study of the system dynamics of software evolution processes: the approach is called *qualitative reasoning*. The approach is based on the observation that neither the empirical data available for study of the dynamics of software evolution processes nor our understanding of the relationships between variables is sufficiently precise as to use traditional quantitative techniques in all their potential. Qualitative reasoning provides advantages which we have discussed in previous papers (see references below). An issue that emerges in all quantitative simulation modelling, particularly in deterministic approaches but also in stochastic approaches, is the need to execute the models as precisely as possible, calling for accurate empirical measures for calibration and model validation. Moreover, classical simulation techniques require precise knowledge of the functional relationships involving the variables being modelled (e.g., software size, effort, defect density and so on). In the majority of software processes, with exception of those at the highest maturity levels, these relationships are likely to be either known incompletely or unknown. Qualitative reasoning, which includes techniques such as qualitative simulation and abstraction, does not solve these problems completely but can ameliorate some of them.

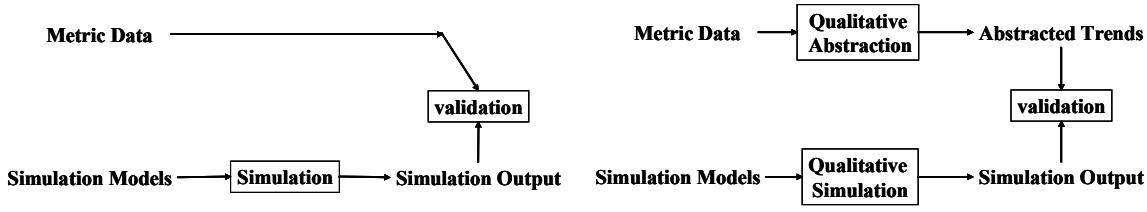
There is an important sub-class of models: the models that can be mathematically represented as a set of differential equations with coefficients expressed as real numbers.

These emerge in many simulation problems including the application of the widely used system dynamics techniques [e.g. Lehman et al 2002]. However, in a qualitative simulation technique such as QSIM [Kuipers 1994], the parameters of the differential equations do not need to be specified as real numbers. It is sufficient to know whether they are positive or negative. Mathematical functions in a QSIM model can be specified simply as monotonically increasing and decreasing functions. That is sufficient for the QSIM simulation engine to explore all the possible behaviours associated to a particular set of differential equations. The output of the simulation is qualitative: it is expressed in terms of shapes and direction, not precise trajectories. For example, a qualitative simulation may be able to tell us under which conditions an important attribute, such as functional growth rate, displays sustained increase (linearly, sub-linearly or super-linearly), steadiness or decrease (again, linearly, sub-linearly, or super-linearly). Table 1 presents the set of symbols we have used for expressing the simulation output [e.g., Janusz & Venkatasubramanian 1991]. The key idea is the abstraction the observed behaviour.

Sign of First Difference dm/dt	Sign of 2nd Difference $\sim d^2m/dt^2$	Symbol	Segment Name
[0]	[0]	—	Constant
[+]	[0]	/	Linear growth
[—]	[0]	\	Linear decrease
[+]	[+]	⌡	Superlinear increase
[+]	[—]	⌠	Sublinear increase
[—]	[—]	⌡	Superlinear decrease
[—]	[+]	⌠	Sublinear decrease
[0]	[+]	∪	Minimum reached
[0]	[—]	∩	Maximum reached

Table 1 – Symbols used to present the qualitative simulation output using QSIM and the corresponding signs for the first and second differences of the variable

One significant difference between qualitative process simulation and the classical counterparts is the need to perform qualitative abstraction (QA) of the empirical data, in order to be able to compare the simulation output to real world data and, in doing so, assess the validity of the simulation model. Qualitative abstraction transforms a sequence of measurements into a pattern, which can be codified as any sequence of the symbols presented in table 1. The simplest QA algorithm obtains the first and second differences of a sequence of measurements and extract the signs of the differences (+,–,0). Next, based on the possible combinations presented in table 1, the algorithm identifies the corresponding sequence of patterns. This difference between traditional quantitative and qualitative approaches is illustrated by the diagrams in figure 1a and 1b below.



Figures 1a and 1b – Model validation for classical quantitative simulation (left) and in qualitative reasoning (right)

3. Empirical Data

The investigation that we are planning to undertake will start with empirical data extraction from code repositories for which we have developed expertise. Previous research has shown that records such as existing change log records, program headers and configuration management offer a limited, but still suitable source in order to study the software evolution. The measures upon which the investigation will be based are obtainable from this type of sources.

As part of an empirical work, data extraction will be performed by using, or adapting, existing tools, as well as creating new tools, specifically created for the extraction, parsing or harvesting the statistical results out of the automatically extracted data. A dual approach is used in this work when referring to collect and analyse data: a discrete approach is defined when a software system is evaluated through the source code of its releases, while a sampling approach is used in presence of a CVS repository, and source code is extracted based on sampling dates (e.g. every week, or month). In this work we plan to extract several attributes for each software system, taking measurements over releases (or sampled systems) of size in number of files, files handled, average complexity and the level of complexity-control work.

We have accumulated experience in the use of a set of metrics to characterise the evolution of software systems at different levels of abstraction: sub-systems, modules (or files or classes) and functions (or procedures). The study of sub-systems have been mainly through the characterization of the evolving folder structure of the source code, which can be seen as a surrogate of the architecture of the system. It is when the behaviour of these is jointly analysed that a richer view of the evolution of a system emerges. Through these visualisations, for example, it has been found, for example, that in the Open Source context, additional stages are likely to emerge in addition to those indicated in models in the literature such as the one in [Rajlich and Bennett 2000]. In particular, they have identified a hibernation stage in some Open Source projects in which for some time no evolution activity is apparent, after which activity is regained. The use of qualitative simulation will help to clarify and put these and other observations into a more solid basis. Access to empirical data is essential in an investigation such as this. A London based software house in the domain of web advertising, actively using Agile processes has provided access to the source code of one of their systems. Moreover, open source projects are important sources of empirical data which we plan also to exploit. There is now in the software evolution community a growing interest and expertise in extracting metrics from these widely available data sources [e.g. RELEASE, Fisher et al 2003]. This offers opportunities to share data-sets and expertise. We also have access to data from at least 3 proprietary systems: the change log records of a large information system in the telecommunication domain, change history records of an operating system kernel and the program headers of a financial transaction system. These data sources provide also input to the proposed research. Even if the proposed investigation focuses on the Agile and open source paradigm and on the already available industrial records, given the increasing popularity of the former and the richness of the latter, the research will generate useful and relevant results and make progress in this area.

4. Research Questions

The general research question to be investigated is the following: what are the characteristics of software evolution under Open Source and Agile paradigms? The detailed research questions to be investigated in our future work include the following:

1. to which degree the empirically observed evolutionary trends in attributes – such as size, structural and functional complexity of software – can be successfully modelled by existing and new qualitative simulation models of the evolution process and
2. what is the support from validated qualitative simulation models to existing empirical generalisations such as the Rajlich and Bennett's staged model of the

software life cycle and the laws of software evolution, and to other alternatives which may emerge?

3. what are the theoretical and practical implications of the answers to questions 1 and 2?

The plan for the investigation of the above questions requires at least the following steps:

a) data extraction from existing records of evolving systems
b) construction and validation of qualitative simulation process models, both generic, and specialised to particular software process domains (e.g. traditional non-agile, agile, open source). The models can be inspired in existing empirical generalisations or in direct observations and insight
c) refinement of the exiting set of empirical generalisations regarding long-term evolution, including Rajlich and Bennett's model, Lehman's laws of software evolution and measures of their empirical support
d) experimentation with the models and systematic analysis of the empirical generalisations, in order to generate a compendium of good practice [e.g. Lehman & Ramil 2001], including recommendations for long-term management of the software process
e) interaction, when possible, with key developers of one collaborator company in order to validate and contrast the output of the data extraction, the empirical generalizations that may emerge from the models

The desired deliverables for our future research will include:

a. a set of qualitative simulation process models specialised to traditional proprietary processes (Waterfall-life processes), Open Source and Agile processes
b. a refined set of empirical generalisations regarding long-term evolution, including refined versions of the Rajlich and Bennett's model, Lehman's laws of software evolution
c. a compendium of good practice, including recommendations for long-term management of the software process
d. an evaluation of existing theory and models of software evolution.

The above represent our long term plans over a research period of two years or so, provided that adequate funding is provided. If no funding is secured, then we plan to still make progress in the topic but a slower rate.

5. Related Work

[Suarez et al 2002] applied qualitative simulation techniques to the software process but with focus on new development, not on long term software evolution as in our approach. The Open Source domain has been studied using traditional direct visualization of trends based on quantitative metric data extracted from a single Open source system [e.g. Capiluppi et al 2004, Capiluppi et al 2005]. To the knowledge of the present authors, our work is the first attempt to analyse the growth trends of a large sample of Open Source and Agile systems using qualitative reasoning techniques.

Exploration of activity levels in software evolution has been the focus of several quantitative simulation models using system dynamics [e.g., Lehman et al 2002]. This modelling effort has been inspired in data and observations of proprietary software. Research efforts specifically involving simulation of some aspects of Open Source development and evolution has also taken place. Examples include [Madey et al 2002], who used the existing SWARM agent-based simulation tools, and [Dalle & David 2004], who built their own agent platform, SimCode. This type of models was used to investigate questions related to the amount of effort allocated to OSS projects and whether a significant attraction of new developers can be achieved or not in the evolution of the project. This research shares with our work the focus on product characteristics (e.g. size and complexity) and on evolution. However their approach is bottom-up, involving rather complex models, while our approach

is top-down, starting with a simple theory or explanation of the phenomenon and much simpler models.

6. Final Remarks

Our previous results in the application of qualitative simulation and qualitative abstraction to empirical data derived from software evolution processes have been already reported in two journal papers [Ramil & Smith 2002; Smith et al 2005] and several workshop presentations [Smith & Ramil 2002, 2003; Smith et al 2004]. Our future work will build upon results achieved, expanding them and extending them to the Agile domain. In the short term (next few months) we plan to continue work in empirical data extraction and in the building of qualitative simulation models for Open Source processes. If we are given the opportunity, we hope to be able to report on our most recent work, including new work we included in [Smith et al 2005], at the coming ProSim 2005 workshop. We welcome the feedback of the community with regards to our plans. We are also currently preparing a proposal to request funds from the UK EPSRC in order to be able to work in the long term goals and deliverables.

7. Acknowledgements

We gratefully acknowledge Dr. Helen Sharp, from the Computing Dept., The Open University, through whom we have been able to get access to the source code of an evolving system, developed under the Agile paradigm.

8. References

[Aoyama 2002] Aoyama M, Metrics and Analysis of Software Architecture Evolution with Discontinuity, Proc. 5th Intl. Workshop on Principles of Software Evolution, IWPSE 2002, Orlando, FL: 103 – 107

[Beck 1999] Beck K (1999); Extreme Programming Explained: Embrace Change, Addison-Wesley, 224 pp.

[Capiluppi 2003] Capiluppi A., Models for the Evolution of OS Projects, Proc. ICSM, Amsterdam, 22 – 26 Sept. 2003, pp. 65 - 74.

[Capiluppi et al 2004] Capiluppi A., Morisio M. & Ramil J.F., Structural Evolution of An Open Source System: A Case Study, Proceedings of the 12th International Workshop on program Comprehension (IWPC), June 24-26, 2004, Bari, Italy, pp. 172 - 182 .

[Capiluppi et al 2004b] Capiluppi A., Morisio M. & Ramil J.F., The evolution of source folder structure in actively evolved open source systems, Proceedings of the 10th International Symposium on Software Metrics, Sept. 11-17, Chicago, Illinois, pp. 2 - 13

[Capiluppi et al 2005] Capiluppi A., Faria A.E. & Ramil J.F., Exploring the Relationship between Cumulative Change and Complexity in an Open Source System, 9th European Conference on Software Maintenance and Reengineering (CSMR), Manchester, UK, March 21-23, 2005

[Dalle & David 2004] Dalle J.M., David P.A. 2004. SimCode: Agent-based Simulation Modelling of Open-Source Software Development, available online at http://opensource.mit.edu/papers/dalledavid2.pdf <as of Feb. 2005>

[FEAST] FEAST, Feedback, Evolution And Software Technology, Dept. of Computing, Imperial College, http://www.doc.ic.ac.uk/~mml/feast/ <as of Feb. 2005>

[Fisher et al 2003] Fischer M, Pinzger M and Gall H (2003); "Populating a Release History Database from Version Control and Bug Tracking Systems", Proc. ICSM 2003, 22-26 Sept, Amsterdam,: 23 – 32

[Glass 2003] Glass RL, 2003, Facts and Fallacies of Software Engineering, Addison Wesley Professional, 224 pp.

[Janusz and Venkatasubramanian 1991] Janusz, M. and V. Venkatasubramanian, (1991) Automatic generation of qualitative description of process trends for fault detection and diagnosis. Engng Applic. Artif. Intell., 4, pp. 329-339.

[Kuipers 1994] Kuipers, B. (1994) Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge, Cambridge, Massachusetts: MIT Press.

[Lehman 1974] Lehman M.M., "Programs, Cities, Students, Limits to Growth?", Inaugural Lecture, in Imperial College of Science and Technology Inaugural Lecture Series, v. 9, 1970, 1974, pp. 211 – 229. Also in Programming approach, Gries D (ed.), Springer Verlag, 1978, pp. 42 – 62. Reprinted as Chapter 7 in [Lehman and Belady 1985].

[Lehman and Belady 1985] Lehman MM and Belady L (1985); Program Evolution – Processes of Software Change, Acad. Press, London, 1985.

[Lehman and Ramil 2001] Lehman MM and Ramil JF (2001); "Rules and Tools for Software Evolution Planning and Management", Annals of Software Engineering, vol. 11, special issue on Software Management, 2001, pp. 15 – 44.

[Lehman et al 2002] Lehman MM, Kahen G and Ramil JF, Behavioural Modelling of Long lived Evolution Processes– Some Issues and an Example, J. of Software Maintenance and Evolution, spec. issue on Separation of Concerns, vol. 14, 2002, pp. 335 – 351

[Madey et al 2002] Madey GR, Freeh VW, Tynan RO. Agent-Based Modeling of Open Source using Swarm, in Proc. of Americas Conference on Information Systems (AMCIS 2002), Dallas, Texas, August.

[Rajlich and Bennett 2000] Rajlich VT and Bennett KH (2000); "A Staged Model for the Software Life Cycle", IEEE Computer, 33(7): 66 – 71.

[Ramil & Smith 2002] Ramil J.F. & Smith N., Qualitative Simulation of Models of Software Evolution, Journal of Software Process: Improvement and Practice, vol. 7, 2002, pp. 95 – 112.

[RELEASE] RELEASE, REsearch Links to Explore and Advance Software Evolution, http://labmol.di.fc.ul.pt/projects/release/ <as of Feb. 2005>

[Smith & Ramil 2002] Smith N. & Ramil J.F., Qualitative Simulation of Software Evolution Processes, Proc. of WESS'02, Oct. 2nd, Montreal, Canada, pp. 41 – 47.

[Smith & Ramil 2003] Smith N. & Ramil J.F., Qualitative Abstraction and Simulation in the Study of Software Evolution, ProSim 2003, Portland, Oregon, May 3-4, 2003.

[Smith et al 2004] Smith N., Capiluppi A., Ramil J.F., Qualitative Analysis and Simulation of Open Source Software Evolution, ProSim 2004, May 24-25, 2004.

[Smith et al 2005] Smith N., Capiluppi A., Ramil J.F., 2005, A Study of Open Source Software Evolution Data using Qualitative Simulation, accepted for publication in the "Special issue on software process modeling and simulation", published by The Software Process Improvement and Practice journal (SPIP).

[Sommerville 2001] Sommerville I, 2001, Software Engineering, 6th Ed., Addison-Wesley, Wokingham, UK, 2001, 720 pp.

[Suarez et al 2002] Suarez AJ, Abad PJ, Gasca RM, Ortega JA, (2002) "Qualitative Simulation of Human Resources Subsystem in Software Development Projects", Proc. 16th Intl. Workshop on Qualitative Reasoning (QR2002), June 10-12, Sitges, Spain, pp. 169 - 176

[Tesoriero and Zelkowitz 1998] Tesoriero R and Zelkowitz MA (1998); "Model of Noisy Software Engineering Data", Proc. ICSE 20, Kyoto, Japan, 19 – 25 April, pp. 461 – 476.

Teaching by Modeling instead of by Models

Thomas Birkhoelzer, Emily Oh Navarro, André van der Hoek

Abstract— **Teaching and training is one of the important applications of software engineering process simulation. Up until this point, however, it has only been used in the context of students running simulations of process models that were built by someone else.**

In this paper, we suggest a different approach: to use the modeling activity for teaching as well, rather than the simulation activity only. In particular, we pro-pose to assign students the task of building a new soft-ware process simulation model using an existing educational software process simulation environment, SimSE.

First experiences from a feasibility project are reported.

Index Terms—**Process modeling, software project simulation, teaching of software engineering.**

I. Introduction

TRAINING and teaching is an important application of software process simulation [1]. Various models and environments have been developed targeting this context, e.g. [3], [4], [5], [6], [7] [8]. These all share the purpose of giving students virtual experiences of realistic software processes that would otherwise be infeasible to practice in an academic environment.

So far, the reported usage of simulation and modeling in this context is always structurally similar: An existing model is used by the trainee for virtual experiences, i.e. simulation is leveraged for teaching, and modeling is done outside the learning situation by an instructor or some other expert beforehand.

From other technical fields, however, it is well known that the modeling by itself provides valuable learning insights. For example, many classes in engineering disciplines include the development of a simulation model in the respective field of application as a final assignment.

In this paper, it is suggested to use software process modeling and simulation in a similar way: the active development of the model as a task for students, not just their passive usage of a pre-existing simulation.

Experiences from a first project are reported and needs for further work and developments are discussed.

Manuscript received February 6, 2005. The SimSE project is partially funded by the National Science Foundation under grant numbers DUE-0341280, CCR-0093489 and IIS-0205724.

T. Birkhoelzer is with the Department of Electrical Engineering and Information Technology, University of Applied Science, Konstanz, Germany (phone: +49 7531 206239; e-mail: birkhoelzer@fh-konstanz.de).

E. Navarro is with the Department of Informatics, Donald Bren School of Information and Computer Science, University of California, Irvine (e-mail: emilyo@ics.uci.edu).

A. v. d. Hoek is with the Department of Informatics, Donald Bren School of Information and Computer Science, University of California, Irvine (e-mail: andre@ics.uci.edu).

II. Didactic goals of modeling

The development of the model by students has three unique didactic advantages:

- *Modeling requires articulateness and explicitness.*
  For a simulation model, all assumed relations and mechanisms of projects or processes must be made explicit and precise in order to be executable. For example, it is one thing to just state that a tool would "improve the process". A simulation model, on the other hand, requires one to articulate this assumption explicitly: which attributes (e.g. error rate or productivity) are influenced and how?
- *Enactment of a simulation provides immediate feedback.*
  Enacting a simulation usually provides immediate and obvious feedback about the consequences of relations and mechanisms stated, much better than any instructor critiques, especially with regard to errors or neglected side-effects. This is not to say that the simulation can replace an instructor. The simulation just provides the mechanical feedback, such that instructors can concentrate on translating this into lessons learned.
- *Creative task as motivation.*
  Technical students – engineers as well as computer scientists – usually love to create things, not just to use them. In this sense, model creation can provide a much higher motivation than just model usage.

III. Modeling a system engineering project

A. Background

As a feasibility study of developing a simulation model by students as part of normal project management course work, three undergraduate students of "project engineering" at the University of Applied Science, Konstanz developed a simulation model as a project assignment. They were in the third year of their studies with a background in electrical engineering and project management, not in software engineering or computer science.

SimSE was chosen as the modeling tool and simulation environment, mainly because of its integrated model builder tool and graphical simulation environment.

Together with the SimSE tools, an existing SimSE model of a software project following a waterfall process model was available, which was used as example and template.

B. SimSE Environment

SimSE is a game-based, graphical, interactive software process modeling and simulation environment designed specifically for generating educational simulations. Its purpose is to allow students to practice quasi-realistic, large-scale software processes in a fun (and hence, more

educationally effective [2]) setting; and to allow instructors to build the simulation models for their students to "play."

One of the most significant features of SimSE is its model builder tool, which was designed to make model building simpler by obliterating the need to learn and program in a process modeling language. The model builder tool completely hides the underlying textual modeling language from the modeler by providing a graphical user interface. This interface allows one to build a model using only buttons, drop-down lists, menus, and text boxes – no programming is required. Once a modeler specifies all of the object types, start state objects, actions, rules, and graphics for a model, the environment then generates a simulation game based on that model.

The graphical, high-level nature of the model input requires less initial learning overhead compared to a special modeling or programming language, which was an important issue considering the students' non-computer science background.

Moreover, the model builder, as well as the generated simulation game, are Java applications. Therefore, only a Java development kit is required as a prerequisite, which eases the usage by students on their private hardware.

More information about SimSE and its waterfall model that was extended for this project can be found at [9].

C. Modeling task

As modeling task, a system engineering project was chosen, i.e. a project combining hardware and software parts into a system. Because SimSE is a game-based simulation environment, it requires that a scenario or "story" accompany each model. In this case, the simulation's goal was the development of a control component for a test robot. The idea of this story stemmed from work done by one of the students during an internship. Thus, this immediately connected the theoretical model to practical experiences.

The students built the model as an extended waterfall model that incorporates hardware aspects as well as software, using the following basic flow: Starting from system requirements, hardware, software, and supply components are pursued in separate paths (with the associated artifacts) and finally integrated into the product.

Therefore, the modeling task was similar to the existing waterfall simulation model. Theoretically, the artifacts, activities, and relations from this existing model just needed to be cloned and extended. At the same time however, to avoid an overly complex model, the existing mechanisms needed to be simplified. Both, appropriate extension and simplification, required a thorough understanding of the model and, more importantly, of the intentions behind the model.

D. Modeling Workflow

The SimSE model builder tool supports a modeling workflow, which closely resembles a didactic decomposition of project management issues.

1. *Definition of the project constituents (artifacts, deliverables, participants, tools).*
   This is the first and basic step of any project management. In the SimSE model builder, this corresponds to defining the object types (templates for the simulation objects) and start state (instantiated objects that the simulation begins with). For the system engineering project, business, software and hardware artifacts were defined reflecting the basic steps of a development process in each category (e.g. specification, design implementation, test). Participants are employees with different experience in these three fields (software, hardware, business).
2. *Definition of the actions a manager (player) can take.*
   These are the inputs into the simulation. There are typically two classes of such actions: task assignments (create, review, correct) and management actions (give bonus, purchase tool, motivate by free coffee, fire). Whereas the first ones are direct consequences of the list of artifacts, the second class enables and enforces the students to define their management style. Each manager/player action corresponds directly to a SimSE action – the specific ways that a player can manage, control, and drive the simulation.
3. *Definition of the actions that occur autonomously.*
   In addition to the actions triggered by the manager, there are events beyond the control of a manager. In SimSE, this is modeled by actions that occur automatically (e.g., employees take breaks) or randomly (e.g., the customer introduces new requirements, employees get sick). The definition of such events teaches basic risk management.
4. *Definition of the effects the actions should have.*
   In SimSE rules are attached to each action. These rules specify how that action affects the rest of the simulation. For example, a creation rule affects the completion percentage of an artifact depending on the productivity of the participant.
   The rules and actions can be prioritized according to which ones should be evaluated first, based on their dependencies. For instance, an action that is triggered based on an employee's energy level (e.g., take a break) should be evaluated after another rule that modifies that employee's energy level is fired.
   Whereas the definition of such rules seems to be straightforward on a first glance, actually conducting it reveals two basic challenges: The large amount of such effects and the difficulty to quantitatively describe it by mathematical formulas. For example, every project manger would immediately agree that the productivity of an employee depends on the experience, the mood, and the number of parallel tasks, but how to model this by a mathematical formula, e.g. as a sum or a product?
5. *Definition of the dependencies.*
   Dependencies between artifacts and activities are modeled in SimSE by effects of rules as well. For example, a creation rule can have the effect of reducing the completion percentage of a dependent artifact (or the effect of increasing the number errors in this artifact).
   Whereas this provides a realistic management situation (a dependent artifact can be worked on before finishing its precursor), it adds another level of complexity. The student project stopped at this point.

6. *Definition of the graphics (in parallel to the previous steps).*
   To each constituent of the project, an image needs to be assigned for the simulation. In addition, a pictorial layout of the simulated office must be defined. Whereas this provides no direct insight into project management, it adds a lot to the impression of ownership.

IV. Lessons and Recommendations

During the project, the following lessons were learned:

- *Modeling is difficult but possible*
  The modeling task provides a challenge to students. The translation of project management knowledge into mathematical formulas and mechanisms is unusual and unfamiliar. Nevertheless, the student group finally succeeded: despite some odds, they were able to form a first simulation model. Of course this model is not yet complete (specifically, further effects could be modeled and many dependencies are missing).
- *Creative aspects provide high motivation*
  The opportunity to create their own processes and mechanisms served as an important motivation for the students. In this context, also less scientific aspects like graphics and playful aspects should not be neglected. It seems to be more fun to design a game-like simulation than just chart outputs.
- *Examples are helpful and necessary*
  For the system engineering project, the most successful modus operandi was the study of the artifacts and mechanisms of the existing model and their appropriate modification. Designing everything from scratch would have been much more difficult. Based on that experience, it is recommended to document and use such examples as kind of templates or patterns.
- *Tools need improvements*
  Within the traditional usage of simulation models in a teaching context, the students use only the simulation environment, not the modeling tool. Therefore, the simulation environments are designed for non-expert use, whereas the usability of the modeling tool got less attention. Instead, the modeling tools were designed to allow maximal expressiveness and flexibility.
  Modeling by students requires modeling tools for non-expert users as well. Usability with minimal training might be more important than complicated functionality.
- *Implementation-independent notational support missing*
  One of the most difficult tasks was the specification of the intended relations and mechanism on paper. For example, the modeling tool provides interactive menus to specify the modeling rules. However, to design, discuss, and document these rules outside the program on paper, an appropriate notation (not a programming language) is necessary. For process modeling, such a notation is not readily available or broadly established. It is difficult for students to develop such a notation by themselves. Therefore, early support by the instructors on formal model design (not just implementation) is necessary. It might even be helpful, if the modeling tools would be accompanied by notational tutorials and examples as well.

V. Conclusion

Modeling by themselves forces the students to be precise and explicit about their assumptions regarding projects and processes, as the enactment by the simulation provides immediate feedback. Moreover, the creative nature of the task seems to meet the motivation of many students in this area.

This is not to say that such modeling should replace the use of preexisting models in a learning context. Both have their distinct advantages: Own modeling is restricted to small problems with a limited level of detail and sophistication and might even remain incomplete. It is not expected that such models can be used to gain really new insights from the model itself. To gain such insights requires using and studying more elaborate and detailed models developed by experts. Using the later one as templates for the own development as described in Chapter 4 can benefit both approaches by improving the own results as well as the appreciation of the "expert-models".

Therefore, it is expected that the development of simulation models by students themselves can be a valuable component of process and project teaching and training complementing the use of expert-models. The overall positive outcome of the system engineering project modeling supports this expectation.

To alleviate this, however, further work is necessary especially towards developing easy-to-use modeling tools, appropriate templates and modeling tasks, and unified and simplified notations.

Acknowledgment

The system engineering project has been developed by Patrick Schnell, Markus Schulz, and Stefan Weidele. We would like to thank them for their intellectual curiosity to try such a task and for their dedication to carrying it through.

References

[1] M. L. Kellner, R. J. Madachy, and D. M. Raffo, "Software Process Modeling and Simulation: Why? What? How"? *Journal of Systems and Software 46*, Elsevier, New York, 1999, pp. 91-105.

[2] M. Ferrari, R. Taylor, and K. VanLehn, "Adapting Work Simulations for Schools," *The Journal of Educational Computing Research, 21(1)*, 1999, pp. 25-53.

[3] A. Drappa and J. Ludewig, "Simulation in Software Engineering Training," *Proceedings of the 22nd International Conference on Software Engineering*, ICSE, Limerick, Ireland, June 2000, pp. 199-208.

[4] J.S. Collofello, "University/Industry Collaboration in Developing a Simulation Based Project Management Training Course," *Proceedings of the Thirteenth Conference on Software Engineering Education and Training*, S. Mengel and P.J. Knoke, Eds.: IEEE Computer Society, 2000, pp. 161-168.

[5] H. Sharp and P. Hall, "An Interactive Multimedia Software House Simulation for Postgraduate Software Engineers," *Proceedings of the 22nd International Conference on Software Engineering*, ACM, 2000, pp. 688-691.

[6] D. Pfahl, M. Klemm, and G. Ruhe, "A CBT Module with Integrated Simulation Component for Software Project Management Education and Training," *Journal of Systems and Software 59*, Elsevier, New York, 2001, pp. 283 298.

[7] E. Oh Navarro and A. van der Hoek, "Software Process Modeling for an Interactive, Graphical, Educational Software Engineering Simulation Game," *Proceedings of the 5th International Workshop on Software Process Simulation and Modelling (ProSim 2004)*, Edinburgh, May 2004, S. 171-176.

[8] Th. Birkhölzer, L. Dantas, C. Dickmann, J. Vaupel, „Interactive Simulation of Software Producing Organization's Operations based on Concepts of CMMI and Balanced Scorecards,“ *Proceedings of the 5th International Workshop on Software Process Simulation and Modelling (ProSim 2004)*, Edinburgh, May 2004, S. 123-132.

[9] E.O. Navarro and A. van der Hoek, “Design and Evaluation of an Educational Software Process Simulation Environment and Associated Model,” *Proceedings of the Eighteenth Conference on Software Engineering Education and Training*, Ottawa, Canada, IEEE, 2005 (to appear).

A Simulation Model for Global Software Development Project

David Raffo and Siri-on Setamanit

Abstract—**Global software development (GSD) is becoming a dominant paradigm in the software industry. Conducting development projects in multiple countries offers many potential benefits including significantly reduced cost and better response times. However, it also poses some difficulties and challenges in managing this kind of project. Software Process Simulation Modeling (SPSM) can be used to support, enrich and evaluate GSD theories, to facilitate understanding, and to support corporate decisions. The discrete-event paradigm and system dynamics paradigm compliment each other and together enable the construction of models that capture both the dynamic nature of project variables and the complex sequences of discrete activities that take place. In this paper, we outline the objectives and the requirements of a GSD model, evaluate different types of simulation paradigms, identify important GSD factors and review relevant quantitative studies and models that could potentially be included in the GSD model. We also present a high-level description of a prototype GSD model we are actively developing.**

Index Terms—**Global Software Development, Distributed Development, Software Process Simulation Modeling, Hybrid Simulation Model**

I. Global Software Development (GSD)

In recent years, global software development (GSD) has become a dominant paradigm in the software industry. Seeing the benefit of low cost and diverse development skill, many software companies have begun to develop software at remote sites or through outsourcing. Other potential benefits include reduction in time-to-market, reduction in development costs, better use of scare resources, and business advantages from proximity to customers [1-4]. Nevertheless, GSD also poses some difficulties and challenges to the development team. Due to geographical dispersion and time-zone differences between development sites, communication and coordination become much more difficult. Cultural and language differences also make it even harder to coordinate and manage collaborative project work. It has been reported that, for work of equal size and complexity, multi-site software development takes much longer than single-site development [5, 6]. Researchers have been attempting to identify and understand the factors that enable and hinder the successfulness of global software development.

Current GSD research may be characterized as exploratory with case studies and other empirical work. Unfortunately, they only address limited aspects of GSD projects. The magnitude and the interaction between important factors that may affect the performance of global software development projects are not well captured or quantified. In short, the overall impact of GSD is still unknown.

This indicates the need to develop a methodology that can integrate and synthesize relevant factors, research, theories, and models in the GSD context, which will enable us to investigate and examine the overall impacts of GSD. We believe that using SPSMs for representing GSD projects is a highly desirable approach for this work. In this paper, we identify GSD model requirements and important GSD factors. We review relevant quantitative studies and models that could potentially be included in the GSD model. Finally, we propose a prototype GSD model that draws upon a number of published GSD empirical studies and incorporates many important GSD factors.

II. Research Objective

The goal of our work is to develop a GSD model which will incorporate diverse research and theories regarding GSD projects (and related fields). It will include a set of relevant empirical factors, development methods, process and product parameters, and project environment factors. The proposed GSD model will fulfill two basic needs:

1. To support, enrich and evaluate GSD theories.
2. To support corporate decisions.

At the practical level, simulation can be used to support and improve organizational decision-making. To support corporate decisions in global software development, software process simulation models should have the capability to address the following questions:

General GSD project:

- Should work be distributed across multiple sites or should it be centralized at single site?
- Under what circumstances do dispersed teams perform better than co-located teams? When should a global software development approach be chosen?
- What are the most important factors in GSD projects?

David Raffo is with College of Engineering and Computer Science and the School of Business Administration, Portland State University, Portland, OR 97207 USA (e-mail: davidr@sba.pdx.edu).

Siri-on Setamanit is with the School of Business Administration, Portland State University, Portland, OR 97207 USA (e-mail: sirion@pdx.edu).

Specific GSD project:

- Which development sites should be included in the project?
- How should work be divided up across sites? What task allocation strategy should be used for a particular project?
- What is the forecasted project performance in terms of cost, quality, and schedule?
- What is the impact of process changes in a GSD context? Should we add process A? Can we minimize or skip a portion of process B?

III. Software Process Simulation Modeling (SPSM)

Software Process Simulation Modeling (SPSM) has been used to address a variety of issues ranging from strategic management, project management planning and control, process improvement, technology adoption, to training and understanding [7]. In addition to the ability of software process simulation modeling to support software project management, it can also be used to support software engineering theory formulation and testing. For example, Abdel-Hamid and Madnick used their system dynamics model to experiment with the applicability of Brooks' Law [8].

Software Process Simulation Models can be used as a platform to combine and synthesize previously developed theories and models, and incorporate a wide range of relevant factors. Thus, simulation models may potentially be used to support real experiments and enrich empirical studies, which will be beneficial in supporting, enriching, and evaluating theories [9, 10].

IV. Comparison of Software Process Simulation Models

Several simulation approaches and languages have been applied to the software development process including knowledge-based simulation, agent-based simulation, system dynamics (or continuous simulation), and discrete-event simulation. Each simulation paradigm has its own advantages and disadvantages. No single simulation approach is the best among all situations. The objective of this section is to identify the most appropriate approach that would be best suited the objective of our study.

A. Knowledge-based Simulation (KBS) Models

Knowledge-based simulation is defined as the application of knowledge-based methods within artificial intelligence (AI) to the field of computer simulation [11]. KBS models view the process from the perspective of the person in the loop or in the system. It is used to describe the personal activities of the people participating in the process. One can see that the level of details provided by KBS would be too complex for the GSD model since the GSD model focuses more on the project level performance.

B. Agent-based Simulation (ABS) Models

Agent-based simulation (ABS) models differ from other simulation models in that they focus on decentralized computation. An agent-based model focuses on individual developers, their characteristics, and how they interact with one another, thus allows one to observe the characteristics of the interactions between the developers and their effects [12].

Like KBS models, ABS models focus on a low level of detail, and seems to be too detailed for the GSD model. However, ABS models could be used as a sub-model to simulate certain social aspects in a software development process such as the interaction or communication between developers. Then, the output from the agent-based model could be used as the input to the GSD model.

C. System Dynamics (SD) Models

SD models describe a system in terms of continuous functions that are solved via numerical integration. The values of the model variables are updated continuously as the simulation proceeds. The system dynamics paradigm is appropriate for exploring the behavior of the systems or variables of interest over time. Its strengths lie in its ability to accommodate the rich interrelationships between variables, particularly the effects of feedback. However, it is difficult to describe process steps or complex sequences of activities using SD Models.

D. Discrete-event Simulation (DES) Models

Discrete-event simulation (DES) models can simulate a step-by-step emulation of the flow of entities through a system. The main advantage of DES models is the ability to capture actual process level details and the ability to represent each work product of the development process as being unique through the use of attributes attached to each work product such as size and complexity. However, the primary limitation of DES models is that they cannot easily represent the dynamic nature of the project variables such as human resources, productivity, defect injection and detection rates, and etc.

E. Hybrid Simulation Models

A hybrid model, as the name suggests, is the model that combines different modeling paradigms to better represent a systems or a software development process. By combining discrete-event and system dynamic simulations, hybrid models can represent entities with attributes being acted upon by activities that are influenced by continuously changing factors [13].

A system dynamics paradigm is suitable for modeling continuous factors and their interactions that may impact GSD projects such as communication, coordination, cultural issues, learning curve, changing staff levels, and dynamically varying productivity. On the other hand, the discrete-event paradigm provides the ability to explicitly represent the process structure and mechanisms used to transfer work products and to coordinate activities.

These two paradigms complement each other and thus provide the capability to capture important issues and factors in a GSD project [14]. Thus a hybrid simulation model combining discrete-event and system dynamics paradigms is the most appropriate approach for developing a GSD model.

V. PROPOSED GSD MODEL

This section can be divided into 3 parts. The first part (Section A) identifies the factors that need to be included in the GSD model. Some of these factors were cited in our previous paper [14]. The list shown below is more comprehensive and it includes factors that we are including in our envisioned full-scale GSD model. In section B, we review several quantitative models and discuss how they can be integrated into the GSD model. Finally, we discuss the major components of our prototype model at a high-level (Section C).

A. Important Factors

In order to effectively represent GSD projects, factors that can affect the performance and the productivity of GSD projects are needed to be included in a GSD model. Factors that are important were identified and placed into three categories, which are (1) the fundamental factors, (2) the strategic factors, and (3) the organizational factors.

1) The fundamental factors:

The fundamental factors are the impact from the characteristics of GSD projects include distance, time-zone differences, and cultural and language differences. A project manager has little or no control over these factors since they are caused by the characteristics of the project itself. However, he can mitigate/alleviate the negative impacts of these factors by using the right strategy and tool support. There are five factors in this category including communication problems, coordination and control problems, cultural differences, language differences, and time-zone differences.

1.1) Communication problems

There are two major communication problems in typical GSD projects: (1) inadequate informal communication and (2) loss of communication richness.

Inadequate informal communication

Informal and unplanned communication is extremely important in supporting collaboration in software development processes [15-17]. However, distance profoundly reduces the amount of informal communication [16-19]. Reduction in the frequency of communication can lead to difficulty in collaborative work, which may lead to longer development cycle times.

Loss of communication richness

Generally, tasks that need coordination and cooperation such as software development tasks require rich communication. The richer the media is, the better the communication. However, distance between development sites often prohibits the use of rich media (e.g. face to face communication, video conferencing, etc.) due to cost, quality, and implementation issues. The loss of communication richness can result in poorer communication between programmers in different sites, which can lead to poorer coordination and cooperation.

1.2) Coordination and control problems

Distance and time-zone differences have negative impacts on coordination and control effectiveness. It is no longer possible to coordinate by a quick phone call or by walking around the office. It is also not possible to control the team members activities by visiting their offices. This can contribute to lower productivity rates and lower software quality, which negatively affect the development cycle time.

1.3) Cultural differences

Many researchers agree that national culture differences is one of the important challenges in global software development [1, 3, 20-23]. Cultural differences can affect a GSD project in different ways, including communication and coordination effectiveness, group decision-making, and team performance. Although there is no general theory on cultural differences, we will distinguish culture based on Geert Hofstede [24] and Edward T. Hall [25] cultural dimensions.

1.4) Language differences

Although programming languages transcend national boundaries, developers still have to communicate in order to clarify things or solve problems. When people communicate in a language other than their own, it is unlikely that they would be able to communicate as effectively as using their own languages. In addition, non-native English speaking people usually prefer asynchronous communication such as email over synchronous communication since it allows them to read and write at their own pace [26, 27]. Unfortunately, asynchronous communication often adds to delays or complicates problem resolution since it can take days to get back-and-forth discussions over email [21].

1.5) Time-zone differences

Time-zone differences can have both positive and negative impact on GSD performance. On the positive side, theoretically, development cycle time could be reduced as much as 50% by using a follow-the-sun development strategy [28]. However, the difference in working hours creates temporal distance between development sites [21], thus inhibits the use of synchronous communication.

2) The strategic factors:

Global software development introduces additional strategic issues (such as development site location, product architecture, and development strategy) that managers have to address in addition to what they are exposed to in managing co-located projects. Their decisions on these issues will have some impact on the performance of GSD projects. Consequently, it is important that these factors be captured in the GSD model. There are five factors in this category, which are development site, product architecture, development strategy, distribution overhead, and distribution effort loss.

2.1) Development site

Different development sites have different technological and communication infrastructures, which can impact communication efficiency. In addition, human resources from different countries will have different characteristics in terms of the level of education, skill, and experience. This can impact productivity rate and quality of the software.

2.2) Product architecture

The product architecture determines whether dispersed sites can work simultaneously and harmoniously. It will affect the selection of the development strategy.

2.3) Development strategy

Development strategy (or task allocation strategy) is concerned with how tasks or work products are allocated across sites. Development strategy will have a direct impact on software development operations since it impacts working hours per day, distribution overhead, and distribution effort loss. There are three fundamental development strategies which are module-based, phase-based, and follow-the-sun strategies [1, 14]. Each strategy will have different impact on project performance [14].

2.4) Distribution overhead

There are two types of distribution overhead: management overhead and transfer overhead. On GSD projects, management also needs extra time to decide which tasks are to be distributed across sites and to monitor the development task since more work will be done at different places in a shorter time [29, 30].

Transfer overhead occurs when the work product is handed over from one site to another [31]. The higher the transfer frequency is, the higher the transfer overhead. This additional overhead impacts effort and duration of the GSD project.

2.5) Distribution effort loss

Distribution effort loss is the additional effort lost due to unfinished tasks [2]. This effort loss can occur when the development includes dependent tasks. For example, coding and testing, if the team from one site cannot finish coding on time, the developers in other development site cannot start testing.

3) Organizational factors:

GSD introduces new forms of development teams -the global team or virtual team. The performance of the team has a strong impact on the performance of the project. The factors in this category are concerned with the impacts from virtual teams, which include team formulation and team dynamics.

3.1) Team formulation

Building relationships and trust between team members are the prerequisite for team formulation. Without trust, it is unlikely that a team will work together effectively since the team members are unwilling to communicate openly across sites. Several studies [1, 20, 23, 32-34] have emphasized the importance of having enough time for members to get to know one another or to have a kick-off meeting at the beginning of the project. Thus, the virtual team that establishes trust between members at the beginning of the project is likely to have better performance.

3.2) Team dynamics

Although trust may have already been established among distributed teams at the beginning of a project, without personal interaction or face-to-face communication, the level of trust will decrease over time [35]. The decrease in the level of trust will negatively affect the team performance. Therefore, it is important that trust be reestablished regularly and continually to maintain the efficiency and effectiveness of a virtual team [1].

B. GSD Quantitative Models

There are several studies and quantitative researches on GSD and related fields. For the initial GSD model, we will focus on main GSD quantitative models including works from Herbsleb et al. [5, 6, 36], Taweel and Brereton [31], and Espinosa and Carmel [37, 38]. We found that these models capture some of the factors mentioned in pervious section. In this section, we will briefly describe these three models and discuss how they can be incorporated into the GSD model.

1) Herbsleb et al.

Herbsleb et al. [5] found that distributed work items appear to take about two and a half times longer to complete than similar items where the work is co-located. Regression and graphical models were developed to identify the mechanisms that contribute to the delay [6]. They found that distance reduces communication frequency, which results in the ineffectiveness of social network and lack of "teamness". This leads to higher number of people involved in distributed work. The number of people involved is a reasonable indicator of coordination issues. Higher number of people suggests less effective communication and coordination, which can lower the productivity of the developer [36].

Contribution to GSD Model: The mechanism of delay helps define the relationship between the communication problem and the coordination problem factors, and how they affect the productivity and performance of the project. This knowledge provides a foundation in modeling the impact of GSD.

2) Taweel and Brereton

Taweel and Brereton [31] develop a mathematical model that represents the relationships between development time (using follow-the-sun strategy) and several factors including development site, time-zone, project resource, and distribution overheads.

Contribution to GSD Model: This model identifies the important factors and their relationship to cycle time that need to be incorporated into the GSD model. The most important contribution is the study of different types of distribution overhead and how they affect the cycle time.

3) Espinosa and Carmel

Espinosa and Carmel developed a mathematical model [37, 38], extended from a coordination model developed by Malone [39, 40], to represent coordination costs due to time separation. The results of the model illustrate the impact of time-zone differences on project cycle time in terms of coordination costs (cost due to delays) and vulnerability costs (costs due to unclear messages).

Contribution to GSD Model: This model emphasizes the importance of time-zone differences. The unique contribution is on how time-zone differences affect coordination through the use of different communication mediums, which can further lead to delays and rework.

One can see that each of these three models capture several important factors mentioned in the previous section, but not all. This indicates the need to have a model that can integrate these models and other important factors together in order to better represent a GSD project.

C. GSD Model Components

As mentioned before, we believe that the ideal SPSM for representing GSD projects would have to effectively support both system dynamics equations and discrete event logic. Thus, our proposed model consists of two major components including continuous model and discrete-event model. The discrete-event (DES) model will represent the actual software development process steps.

The DES section can be divided into several sub-models depending on the number of development sites. In other words, each development site has its own DES sub-model. Each development site may have different process steps depending on how tasks are allocated and the activities needed to be performed. This allows us to capture the impact of development strategy on performance. Difference in time-zones can also be modeled by having different site operates at different working hours. An artifact or work product will actually be passed between sites at the end of working day in order to capture the effect of distribution overhead and distribution effort loss.

The continuous portion of the model will include one global continuous sub-model as well as a site-specific sub-model for each development site. The global sub-model captures the overall project environment which includes planning and controlling activities. The information about the project progress will come from the DES model. The global continuous sub-model compares the project progress to the plan and adjusts the plan as necessary.

On the other hand, each development site will have its own site-specific continuous sub-model. The site-specific sub-model represents aspects that may be different between development sites including human resource, productivity, and error generation and detection rates. The productivity and the number of developers available will be sent to the corresponding DES sub-model to drive the development activities.

The factors associated with communication problems, coordination problems, cultural and language differences will be applied to the productivity rate before sending this rate to the DES sub-model in order to capture the impact of GSD. The picture of the prototype model that we currently develop is shown in figure 1 on the next page.

The discrete-event paradigm and continuous paradigm compliment each other and together enable the construction of models that capture both the dynamic nature of project variables as well as the complex sequences of discrete activities that take place. This combined model will allow a series of discrete process steps to be executed in a continuously varying project environment, which allow us to better represent the GSD project.

VI. Conclusion

In conclusion, software process simulation models can be used to support empirical studies and experiments in a GSD context. Using SPSMs will speed the development of knowledge about GSD. This will help in evaluating, supporting, and enriching theories about GSD. With the knowledge and understanding at the theoretical level, SPSMs can then be used to support and improve corporate decisions by addressing practical questions about GSD in a broad or project-specific manner. We believe that the hybrid simulation model combining continuous and discrete-event paradigms will be an ideal platform for constructing a GSD model. In this paper, we identify the factors that need to be included in the model, review the quantitative model that can be integrated to the GSD model, and provide an overview of the components we will be including in our prototype GSD model.

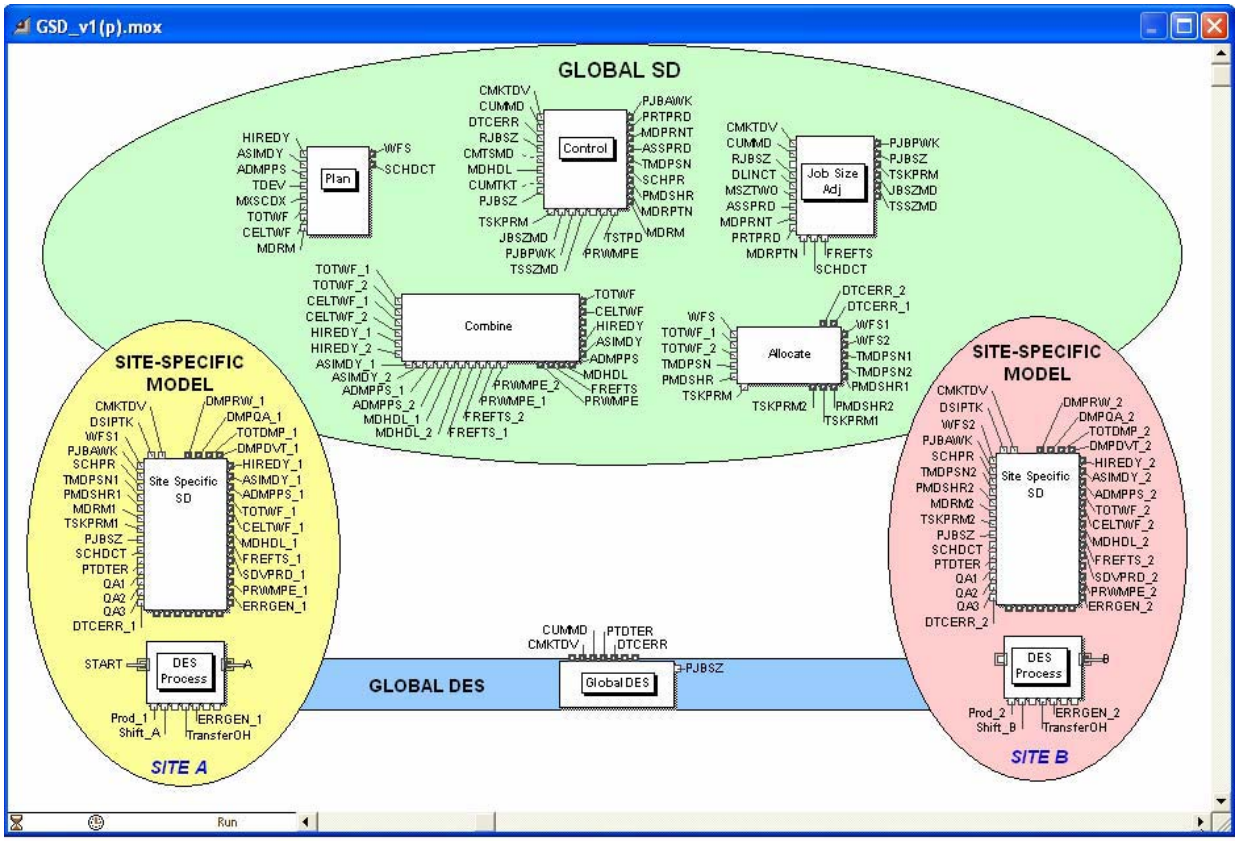


Figure 1: GSD Prototype Model (high-level view)

REFERENCES

[1] E. Carmel, *Global Software Teams*. Upper Saddle River, NJ: Prentice Hall PTR, 1999.

[2] I. Gorton and S. Motwani, "Issues in Co-operative Software Engineering Using Globally Distributed Teams," *Information and SoftwareTechnology*, vol. 38, pp. 647-655, 1996.

[3] J. D. Herbsleb and D. Moitra, "Global Software Development," *IEEE Software*, pp. 16-20, 2001.

[4] J. Norbjerg, E. C. Havn, and J. P. Bansler, "Global Production: The Case of Offshore Programming," presented at Wirtschaftsinformatik '97, Physica-Verlag, Berlin, 1997.

[5] J. D. Herbsleb, R. E. Grinter, and T. A. Finholt, "An Empirical Study of Global Software Development: Distance and Speed," presented at ICSE 2001, Toronto, Canada, 2001.

[6] J. D. Herbsleb and A. Mockus, "An Empirical Study of Speed and Communication in Globally Distributed Software Development," *IEEE Transactions on Software Engineering*, vol. 29, pp. 481-494, 2003.

[7] M. I. Kellner, R. J. Madachy, and D. M. Raffo, "Software Process Simulation Modeling: Why? What? How?," *Journal of Systems and Software*, vol. 46, 1999.

[8] T. Abdel-Hamid and S. Madnick, *Software Project Dynamics: An Integrated Approach*: Prentice-Hall, 1991.

[9] J. Munch, D. Rombach, and I. Rus, "Creating an Advanced Software Engineering Laboratory by Combining Empirical Studies with Process Simulation," presented at The International Workshop on Software Process Simulation and Modeling, Portland, OR, USA, 2003.

[10] I. Rus, S. Biffl, and M. Halling, "Systematically Combining Process Simulation and Empirical Data in Support of Decision Analysis in Software Development," presented at The fourteenth International Conference on Software Engineering and Knowledge Engineering (SEKE'02), Ischia, Italy, 2002.

[11] P. A. Fishwick and R. B. Modjeski, *Knowledge-Based Simulation: Methodology and Application*, vol. 4. New York: Springer-Verlag New York, Inc., 1991.

[12] T. Wickenberg and P. Davidsson, "On Multi Agent Based Simulation of Software Development Processes," presented at Multi-Agent-Based Simulation, Third International Workshop, MABS 2002, Bologna, Italy, 2002.

[13] W. Wakeland, R. Martin, and D. M. Raffo, "Using Design of Experiments, Sensitivity Analysis, and Hybrid Simulation to Evaluate Changes to a Software Development Process: A Case Study," presented at The International Workshop on Software Process Simulation and Modeling, Portland, OR, USA, 2003.

[14] D. M. Raffo, S. Setamanit, and W. Wakeland, "Towards a Software Process Simulation Model of Globally Distributed Software Development Projects," presented at The International Workshop on Software Process Simulation and Modeling, Portland, OR, USA, 2003.

[15] B. Curtis, H. Krasner, and N. Iscoe, "A Field Study of the Software Design Process for Large Systems," *Communications of the ACM*, vol. 31, pp. 1268-1287, 1988.

[16] R. E. Kraut and L. A. Streeter, "Coordination in Software Development," *Communications of the ACM*, vol. 38, pp. 69-81, 1995.

[17] J. D. Herbsleb and R. E. Grinter, "Splitting the Organization and Integrating the Code: Conway's Law Revisited," presented at International Conference on Software Engineering (ICSE'99), Los Angeles, CA, 1999.

[18] J. D. Herbsleb and R. E. Grinter, "Conceptual Simplicity Meets Organizational Complexity: Case Study of a Corporate Metrics Program," presented at International Conference on Software Engineering, Kyoto, Japan, 1998.

[19] T. J. Allen, *Managing the Flow of Technology*. Cambridge, MA: MIT press, 1977.

[20] D. W. Karolak, *Global Software Development: Managing Virtual Teams and Environments*. Los Alamitos, CA: IEEE Computer Society, 1998.

[21] E. Carmel and R. Agarwal, "Tactical Approached for Alleviating Distance in Global Software Development," *IEEE Software*, pp. 22-29, 2001.

[22] R. D. Battin, et al., "Leveraging Resources in Global Software Development," *IEEE Software*, pp. 70-77, 2001.

[23] D. E. Damian and D. Zowghi, "The Impact of Stakeholders' Geographical Distribution on Managing Requirements in a Multi-site Organization," presented at IEEE Joint International Conference on Requirements Engineering, Essen, Germany, 2002.

[24] G. Hofstede, *Culture's Consequences: International Differences in Work-Related Values*. Newbury Park, CA: Sage Publications, Inc., 1984.

[25] E. T. Hall, *Beyond Culture*. New York, NY: Doubleday Books, 1976.

[26] L. Keil and P. Eng., "Experiences in Distributed Development: A Case Study," presented at The International Workshop on Global Software Development, Portland, OR USA, 2003.

[27] H. Ishii, "Cross-Cultural Communication and CSCW," in *Global Networks: Computers and International Communication*, L. M. Harasim, Ed. Cambridge, MA: MIT Press, 1993, pp. 143-151.

[28] E. Carmel, "The explosion of global software teams," in *Computerworld*, 1997.

[29] G. H. Anthes, "Software Development Goes Global," in *Computerworld*, 2000.

[30] R. Fournier, "Teamwork is the Key to Remote Development," in *Inforworld*, 2001.

[31] A. Taweel and O. P. Brereton, "Developing Software Across Time Zones: An Exploratory Empirical Study," presented at Informatica, Ljubljana, Slovenia, 2002.

[32] D. E. Perry, G. S. Gil, and L. G. Votta, "A Case Study of Successful Geographically Separated Teamwork."

[33] E. Harding, "US Companies Finding that CASE Travels Well in India -- Surplus of Skilled Software Professional Makes Outsourceing, Joint Projects Attractive," in *Software Magazine*, vol. 11, 1991, pp. 24-28.

[34] B. Geber, "Virtual Teams," *Training*, vol. 34, pp. 36-40, 1995.

[35] D. Meyer, "A. Tech talk: how managers are stimulating global R&D communication," *Sloan Management Review*, 1991.

[36] J. D. Herbsleb and A. Mockus, "Formulation and Preliminary Test of an Empirical Theory of Coordination in Software Engineering," presented at the ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE), Helsinki, Finland, 2003.

[37] J. A. Espinosa and E. Carmel, "Modeling Coordination Costs Due to TIme separation in Global Software Teams," presented at The International Workshop on Global Software Development, Portland, OR USA, 2003.

[38] J. A. Espinosa and E. Carmel, "Modeling the Effect of Time Separation on Coordination Costs in Global Software Teams," presented at The 37th Hawaii International Conference on System Sciences, Hawaii, USA, 2003.

[39] T. W. Malone, "Modeling Coordination in Organization and Markets," *Management Science*, vol. 33, pp. 1317-1332, 1987.

[40] T. W. Malone and S. A. Smith, "Modeling the Performance of Organization Structures," *Operations Research*, vol. 36, pp. 421-436, 1988.

Author Index