Building a High-Level Process Model for Soliciting Requirements on Software Tools to Support Software Development: Experience Report

Ilia Bider[1,2], Athanasios Karapantelakis[3], Nirjal Khadka[1]

[1] Department of Computer and System Science (DSV) of Stockholm University, Kista, Sweden
[2] IbisSoft AB, Stockholm, Sweden
[3] Ericsson Research, Kista, Sweden
ilia@{ibissoft|dsv.su}.se, athanasios.karapantelakis@ericsson.com, nirjal@gmail.com

Abstract. Use of software tools to support business processes is both a possibility and necessity for both large and small enterprises of today. Given the variety of tools on the market, the question of how to choose the right tools for the process in question or analyze the suitability of the tools already employed arises. The paper presents an experience report of using a high-level business process model for analyzing software tools suitability at a large ICT organization that recently transitioned to scrum-based project methodology of software development. The paper gives overview of the modeling method used, describes the organizational context, presents a model built, and discusses preliminary findings based on the analysis of the model.

Keywords: Business process, requirements elicitation, software development, Scrum, project management, tool support, business process modeling.

1 Introduction

This paper is an experience report from a project that had both practical and research objectives. The project was conducted at a large ICT organization that recently went through reengineering of their software development process. The reengineering concerned transforming this process:

- from a traditional phase-based development approach with local software development teams
- to working in an iterative manner using the Scrum project management methodology and employing geographically distributed teams that have cultural differences and may work in different time zones.

The following two practical objectives were identified for our project:

1. Gain better understanding of current software development practice and map this understanding to a model

2. Use the model to analyze the suitability of project structure and software tools currently supporting the development process and find potential improvements both in the structure as well as the tools themselves.

For completing these tasks, we used a business process modeling technique from [1]. A model build with this technique is expressed in terms of a small number of steps (phases) of the process and relationships of different kinds between these steps. Some of these relationships are well known, e.g. input/output relationships [2]; others are new, like weak dependencies and team intersections. [1] suggests using such a model for choosing an appropriate cloud service to support the process in question. In this paper, we apply the model from [1] for different purpose, see 2 above. In connection to using modeling technique from [1], the following two research objectives were identified for the project:

1. Test the technique suggested in [1]. In particular, we wanted to see whether a high-level process model suggested can be built for a relatively complex process.
2. Test whether such a model can be used for a different purpose than originally suggested.

The rest of the paper is structured in the following way. In Section 2, we describe the project and its organizational context. In section 3, we give a brief overview of the framework suggested in [1]. In Section 4, we present a high-level model of the software development process built based on the framework. In Section 5, 6 and 7, we analyze tools used for supporting the process and give our suggestions for improvements. Section 7 summarizes the findings and draws plans for the future.

2 The project

In order to achieve our objectives as outlined in Section 1, we studied the software development process in one of the Product Development Units (PDU) of a large ICT organization. The product under development was a complex, commercial-grade telephony switching software solution. The organization in question has recently transitioned from a waterfall-like, phase-based software development approach to more flexible iterative development process, using the Scrum framework for project management. Given the novelty of this transition to a new development approach, there was vested interest in potential findings of our study not only from us researchers, but also from project managers from the development unit, who were interested in using our results to enhance the performance of the development process for future product development.

In order to gain understanding on how the development process worked, we started with brainstorming sessions between the members of the research team, without involving external stakeholders. Our research team had access to relevant documentation on the development process such as project plans, process workflow charts, and software development tools documentation.

The results of the aforementioned brainstorming sessions allowed us to form an initial hypothesis around how the process worked. We formalized this hypothesis into a mental model, focusing on identifying the actors participating in the development process and the interactions between them. The intention was to use this hypothetical model as support to our discussions with representatives from the development units. Through those discussions, the model would be refined and finalized. Table 1 list and describes the different actors in the development process.

Table 1. Description of actors and job roles in the product development

Actor	Description	Job Role
Project Management (PM)	Project management is responsible for coordinating geographically distributed software development teams by ensuring that each team's deliverables are finished on time. A sub-project manager is responsible for communicating requirements from project management to the leader of each team, and receiving feedback on the team's progress from the team leader. A Total Project Manager holds regular meetings with sub-project managers to stay informed on the progress of individual teams.	Total Project Manager, Sub-project managers
System Management (SM)	System management has a broad technical perspective of the product under development and is responsible for extending the business requirements drafted from external stakeholders to technical requirements to be used by software development teams.	System Manager
Feature Team X (FT-X)	Feature teams are geographically distributed software development teams. Each team develops a specific part of the product (one "feature"). A team leader is responsible for facilitating communication between the feature team and a representative from PM.	Scrum Master, SCRUM team member
Release Management (RM)	Release management is responsible for integrating all features delivered from the feature teams into a complete product and authorizes the software for commercial release by doing quality assurance ("acceptance testing").	Quality Assurance (QA) Manager, QA-team

Using data from Table 1, we identified a number of individuals representing different actors and interviewed them. The interview questions concerned the type and quality of interactions of the actor being interviewed with other actors, in order to identify potential issues hindering the efficiency of these interactions (for example a sub-optimal performance of a tool leading to inefficient communication).

3 Overview of the high-level business process modeling

According to [1], a high-level business process model consists of the following elements (syntactical units of the modeling language):

- Steps that represent work-packages to be completed in the process, each step having a unique name.

- Relationships between the steps. Relationships are typed and there can be more than one relationship between a pair of process steps, different relationships belonging to different types. Depending on the type, relationships can be symmetrical, or asymmetrical. In case of asymmetrical relationships there can exist up to two relationships for a given pair of steps. An example of an asymmetrical relationship between the steps is the input/output relationship. If a relationship exists between steps A, B it means that there is formalized output from step A that is used as input for step B, but not the opposite.
- We introduce two ways of depicting a model for a particular business process, namely using a graphic notation or relationships matrices.
- In the graphical form, the model consists of a set of diagrams, one for each type of relationships. In a diagram, steps are represented as rectangles (boxes) that have step names as labels inside them. Relationships are represented as lines between the step boxes.
- In the matrix form, the model consists of a set of square matrices, one for each type of relationships, where both columns and rows correspond to the process steps. Intersection between a row and a column in a matrix shows a relationship between the two steps. The type of content in the cells depends on the relationships type.

In this paper, we mostly use the matrix form for depicting a model, as it is easier to work with matrices in a formal way. Some relationships are basic, i.e. they are determined when building a model. Other relationships are derived to be used for determining requirements on tools to support the process. A derived relationships matrix is obtained via transforming a matrix of one of the basic relationships, or via merging two or more other matrixes.

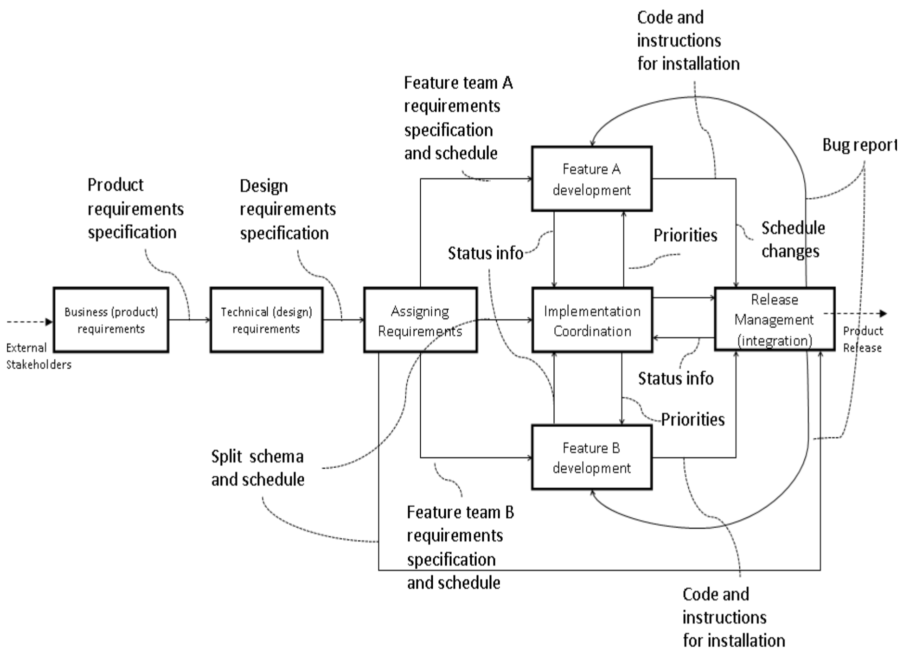
We assume that the number of steps chosen for building a model is rather small (under 10, preferably 5 or 6), so that the whole model is compact in both its matrix and graphical form. Having a small number of steps means that each step represents a rather large work package, which on its own can be split in smaller steps later.

A detailed description of the relationship matrices, which constitute the semantics of our modeling language, is presented in the next session wherein we describe the model built for our case.

4 The model described

4.1 Steps

We identify seven steps in the PDU software development process as illustrated in Fig. 1 (together with input/output relationships, see section 4.2). The central and most complex part of the diagram in Fig. 1 is the development of separate software modules (part of the total software product, also known as “features”) and their integration coordinated by project management. Due to the size and complexity of the product, software development is split to parts and assigned to different teams across the world. These parts are also known as product features, representing a subset of the functionality of the complete product. Each team delivers their own feature; hence the teams are given the name “feature teams”. Feature teams periodically deliver their part of functionality to Release Management for integration with other features. Note that in some cases, there exist inter-dependencies of features; in this case, a feature developed by one team may depend on a feature developed by another team. In such cases, one team has to wait for the other to deliver, which can cause delays to the release schedule (see section 4.2).



Code and instructions for installation
Feature team A requirements specification and schedule
Bug report
Feature A development
Product requirements specification
Design requirements specification
Status info
Priorities
Schedule changes
Business (product) requirements
Technical (design) requirements
Assigning Requirements
Implementation Coordination
Release Management (integration)
External Stakeholders
Product Release
Status info
Priorities
Split schema and schedule
Feature B development
Feature team B requirements specification and schedule
Code and instructions for installation

Fig. 1. Software development steps and input/output relationships (partial view).

For reasons of simplifying the model (see Fig. 1), we present only two feature development steps (features A and B, developed by feature team A and B respectively). In a realistic scenario, there can be more than two such steps; however, we consider the abstraction to two steps sufficient to capture feature teams interdependencies, wherein feature B depends on feature A. The rest of this subsection describes the steps in our model in greater details:

1. BR: Business *(product) requirements* – The business requirements are created as a result of a process involving identification of a business opportunity, discussion on its commercial viability, and ultimately drafting of a formal business requirements document stored in a product management tool.
2. TR: *Technical (design) requirements* – business requirements are converted into design requirements using the expertise of the System Management group (see Table 1). The outcome is a list of technical requirements stored in a requirements management tool. Typically, from one high-level business requirement, a number of technical requirements are created.

3. AR: *Assigning requirements* to feature development teams – the technical requirements are analyzed and their implementation is assigned to various teams. The process of analysis and assignment is done in meetings with participants from the feature teams, as well as project and system management. At the same time the schedule of delivery to Release Management is designed, with the intention to meet the deadlines set by the project timeline.
4. FD-A: *Feature A development* – this step represents the internal process of a team developing feature A. In case of feature dependencies, team A has to deliver both to the Release Management team for integration with other deliverables, but also to team B – for integration with their own deliverable, which depends on feature A. Feature team A in this case would receive bug reports – if any – on their deliverable from both team B as well as the release management team.
5. FD-B: *Feature B development* - this step represents the internal process of a team developing feature B. In this case, the team expects to receive some part of functionality of feature A from feature team A before it can deliver to integration. There is an open line of communication via a bug reporting system towards feature team A in case bugs are discovered.
6. *IC: Implementation Coordination* – This step contains the activities of project management, which is responsible for coordinating development among feature teams and the release management team. Project management receives feedback on the development progress from the feature teams and communicates task prioritization based on the current project status and needs. Project management therefore ensures that release management receives the software deliverables from feature teams in time and maintains an open line of communication with release management, which reports back on whether the integrated feature set meets their quality requirements or there are any residual defects (e.g. bugs, missing functionality) found in one or more of the deliverables.
7. *RM: Release management (integration)* - In this step, a set of engineers forming the RM team integrates all deliverables from the feature teams. After integration, the RM team tests the product feature set in a process known as acceptance testing. If integration is successful and acceptance testing meets the quality criteria of the RM team, then the integrated feature set can be released to the customer (in what is known as Product Release – PR). In any other case, the product release schedule towards the customer is delayed and the issues observed in acceptance testing are submitted through an online bug tracking system to the feature teams (project management is also notified).

4.2 Input/output relationships in detail

Input/output relationships show dependencies of one step on the results achieved in another. In graphical form, the dependencies are represented as arrows between the steps, where the text attached to an arrow explains the nature of the output from one step serving as an input into another (see Fig. 1, 2).

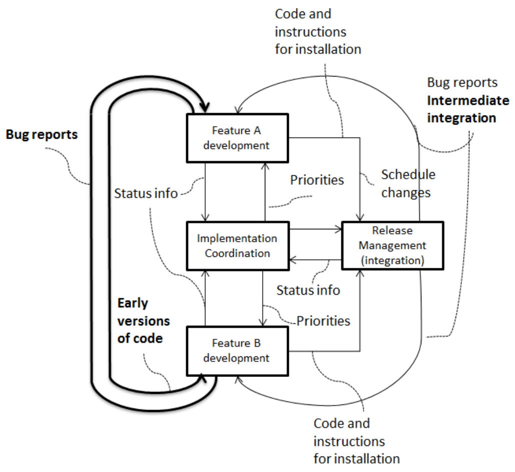


Fig. 2. Detailed view of the input/output relationships between the last 4 steps of the software development process under study. Relationships missing from Fig.1, are presented in bold. They show a feature inter-dependency scenario, where feature B is dependent on feature A.

Table 2. Input/output relationships as a matrix

Output Input	**BR**	**TR**	**AR**	**FD-A**	**D-B**	**IC**	**RM**
BR							
TR	*Product req. spec.						
AR		*Design req. spec.					
FD-A			*Feature A req. spec. ,Schedule			Priorities	Bug reports, Preliminary integration.
FD-B			*Feature B req. spec., Schedule	Early versions of code		Priorities	Bug reports, Preliminary integration
IC			*Split schema, Schedule	Status info	Status info		Status info
RM			*Split schema, Schedule	*Code and instruct.	*Code and instruct.	Schedule changes	

In matrix form, input/output relationships are shown in the following fashion: Cell (a,b) in the matrix, where a refers to a column and b to a row, specifies what result (i.e., output) from column step a (if any) is used as input to row step b. In addition to the name of result, a cell can be marked with asterisk (*) indicating that the result is required for step b to be started the first time. The input/output matrix as shown in Table 2, represents formalized casual relationships between the steps that are analogues to input/output connections in IDEF0 specification [2]. In addition to the formalized relationships, there could be informal input/output relationships between the steps, which are explained in Section 4.4.

4.3 Parallel execution and dependencies

The *parallel execution* matrix shows whether two steps are allowed to be executed in parallel. If ongoing activity inside step *a* does not forbid carrying out activity in step *b*, then both cells (*a*,*b*) and (*b*,*a*) are marked with *x* (the matrix is symmetrical). If none of the steps can run in parallel, the parallel execution matrix will be empty. This would be the case if the system development process in our example was carried out in successive phases (e.g. waterfall model). In our case, *Business requirements*, *Technical requirments* and *Assigning requirements* are executed in the sequential fashion, while *Feature A,B dvelopment*, *Implementation coordination* and *Release management* run in parellel. Due to the lack of space, we do not show this matrix.

[1] suggests combining the *input-output* matrix with *parallel execution* matrix to get a new view on complexity of the process. Table 3 is produced by merging Tables 2 and a corresponding parallel execution matrix according to a simple rule: cell(*a*,*b*) get crossed in the new table only if the cell is non-empty in both input-output matrix and parallel execution matrix. As only the last four steps are completed in parallel, we show in Table 4 only dependencies between the last four steps (all other cells will be empty. We will refer to the merged matrix as to *parallel dependencies* matrix. The cross in cell(*a*,*b*) in this matrix means that steps *a* and *b* can run in parallel at the same time as *b* is dependent on results from *a*.

The parallel dependencies matrix shows the potentially weak points in the process that require special consideration, otherwise the process will not run smoothly.

Table 3. Parallel dependencies between the last four steps of the model

	FD-A	**FD-B**	**IC**	**RM**
FD-A			x	x
FD-B	x		x	x
IC	x	x		x
RM	x	x	x	

4.4 Weak dependencies

Weak dependencies show whether one step might require information from another step that is not part of the relationships formalized in the *input/output* matrix. For example, IC may want to inspect the source code of a troublesome module to determine the severity of the problem when re-scheduling delivery. Cell (*a*,*b*) in this matrix specifies what kind of information from step *a* might be needed to complete step *b*. A part of the weak dependencies matrix for our systems development process is given in Table 4. This part concerns the most troublesome spots in the development process – the one that have parallel dependencies.

The concept of *weak dependencies* reflects the needs for informal communication in the frame of a process instance. It is not always possible to include everything that might be needed for the next step in the formal results, as different instances might require completely different information from the previous steps. It is better to start looking for this information on the demand basis, i.e., when there is a need for it.

Table 4. Weak dependencies

	FD-A	**FD-B**	**IC**	**RM**
FD-A		Explanations of code behavior		Explanation of test results/bug reports
FD-B	Details on the status Explanations of test results/bug reports			Explanations of test results/bug reports
IC	Details on the status	Details on the status		Details on the status
RM	Explanations of code behavior	Explanations of code behavior		

4.5 Teams and their relationships

The *teams* matrix shows the presence of collaborative teams and their relationships. The presence of teams is shown in the diagonal of the *team*s matrix :cell (*a*,*a*) is marked with the light gray color if the team for step *a* consists of more than one person. The non-diagonal elements show whether the teams participating in different steps intersect. If the teams for steps *a* and *b* intersect but not coincide, we mark both cells (*a*,*b*) and (*b*,*a*) with the light gray color. If the teams coincide, we mark these cells with the dark gray color. We also use the lighter gray color if the intersection is very “thin”. In our case, all steps have teams, and many teams intersect. The composition of the teams in terms of Table 1 is as follows (due to the lack of space, we do not present the matrix itself):

- BR team = External stakeholders (see Section 4.1) + SM
- TR team = SM + PM + representatives of FT-X, and RM
- AR team = SM + PM + representatives of FT-X, and RM
- FD-A team (FD-B has the same structure) = FT-X + Subproject-manager from PM
- IC team = PM

- RM team = RM

By merging the *weak dependencies* matrix with the *teams* matrix (Section 4.5), we get a view on the needs for inter-step collaboration. The part of the merged matrix for the last four steps is shown in Table 5. The cells with weak dependencies but without grey background warrant special attention when considering tool support. They mean there are needs for informal exchange, while teams do not intersect. Even the non-empty cells with the background color very light may require special attention as intersection may not be strong enough to serve as the only channel for informal exchange. As can be seen from Table 5, in our process, there are a number of empty cells with the white background, or only a slightly gray one. This requires special attention when analyzing the tools used for process support.

Table 5. Weak dependencies merged with Teams matrix

	FD-A	**FD-B**	**IC**	**RM**
FD-A		Explanations of code behavior		Explanation of test results/bug reports
FD-B	Details on the status Explanations of test results/bug reports			Explanations of test results/bug reports
IC	Details on the status	Details on the status		Details on the status
RM	Explanations of code behavior	Explanations of code behavior		

5 Requirements on tool support

Our previous research on business process support services lists a number of capabilities to be expected from process support tools [1]. In this paper, we will consider only three of them, namely: (1) *Information Logistics Support* (*ILS*), aimed at providing process participants with all information they need to complete their work without being overwhelmed by the details that are not relevant; (2) *intra-step collaboration support* aimed at providing a team working on the same step with means to store/retrieve intermediate results and communicate internally synchronously and/or asynchronously; (3) *inter-step collaboration support* for providing the teams, or individuals working on different steps with means to access intermediate results obtained in each other's steps and communicate between them synchronously and/or asynchronously.

ILS is important to have when there are input/output dependencies between the steps the teams of which do not coincide and can be provided in two ways: (a) by sending the results to the next step team, e.g., via email; (b) by providing a shared space where the results are stored and made available for the participants of the next step. The second type of logistics is more appropriate when there are loops in input/output flow and/or there are parallel dependencies. Loops can be detected via analysis of the input/output matrix (see Section 4.2). Presence of loops is identified by symmetrical non-empty elements in this matrix (or a derived matrix with transitive input/output relationships [1]).

In our case, Table 2 shows several loops (they are also visible in Fig. 1 and 2). Presence of loops indicates that the same input can be provided several times which is best handled by a shared space with version control. Parallel dependencies identify that input can be sent in portions, and the next portion can negate what has been sent in the previous one. When such disrupted input is expected, prompt notification of the new portion arrival is needed. This can be arranged via shared spaces supplemented by notification mechanisms. Table 2 shows a number of parallel dependencies which should be taken care of by process support tools.

Intra-step collaboration support is required when there is a team in at least one step. The presence of step teams can be seen in the *teams matrix* (see Section 4.5). As was mention in Section 4.5 there are teams for each step identified in our process, which warrants the needs for intra-step collaboration support.

Inter-step collaboration support is required when there is informal information exchange between the steps the teams of which do not intersect, or their intersection is too “thin”. These situations can be identified by analysis of the derived matrix acquired by merging the *teams matrix* with the *weak dependencies* matrix, see Section 4.5. Table 5 includes non-empty elements with white background (no intersection) or very light background (“thin” intersection), showing importance of *intra-step collaboration support* for our process.

6 Tools in use

General Purpose Tools (GPT) such as email, word/spreadsheet processors and instant messaging are used when necessary while the specialized tools listed below have specific functions:

- AB – Automated Build, continuous integration tool for building snapshots of code
- BR – Bug Tracker, an in-house developed bug reporting tool
- DE – Development Environment for authoring code.
- FT – Tool for function and unit test - automated test framework
- FS – Secure file server, for feature teams to deliver code to Release Management for integration or to other feature teams in case of feature inter-dependencies.
- PPM – Product and Portfolio Management tool for project management.
- RC – Requirements Composer for defining and following up the requirements.
- ST – Automated System Testing tool for regression testing
- TM – Test Management Tool for documenting test cases
- ST – Scrum Tool for backlog management.
- RP – Version control system, for collaborative code development in feature teams.

The mapping of tools to teams that use them is summarized in Table 6. The diagonal shows which tools are used in each step, other cells show the tools used for transferring input/output (Table 2) or as channels for weak dependencies between the steps. Thus, the non-diagonal part of Table 6 comprises two matrixes: one describes tools used for *ILS* (input/output), while the other is related to *intra-step collaboration* (weak dependencies).

Table 6. Tool usage

	BR	**TR**	**AR**	**FD-A**	**FD-B**	**IC**	**RM**
BR	PPM						
TR	PPM,GPT	RC,PPM					
AR		RC	RC,PPM				
FD-A			RC	DE,FT,FS, RP,ST,TM	GPT	GPT	BR,FS
FD-B			RC	FS,BR,GPT	DE,FT,FS, RP,ST,TM	GPT	BR,FS
IC			RC	GPT	GPT	RC,PPM	BR,GPT
RM			RC	FS, GPT	FS,GPT	GPT	AB,BR,ST,TM

7 Analysis of tools in use

Through analysis of tools, we found lack of support for weak dependencies between IC on one hand, and features teams and RM on the other (see the IC-row in Table 5). In order to learn about the status of the features development, project management manually checks with each feature team their progress (i.e. through e-mail or phone conversations), and records information on it in a spreadsheet. Because of the subjective nature of the progress reports, the project manager we have interviewed has to invest additional time to keep track of progress for some of the teams. To improve support for weak dependencies, a tool that provides integrated picture of the teams activities seems necessary. Such a tool could analyze activities around different requirements assigned to a team, e.g. code compilations, bug reports, etc. and give warnings on suspicious too much activity or luck of such.

8 Concluding remarks

In this paper, we applied the theoretical framework for selecting software tools for business process support from [1] to a real case of software development process in a multinational telecommunications provider. From our analysis, we found that the software development process was missing a tool for communicating the progress of software development from the features teams to Implementation Coordination which could result in delayed release schedules. In this way, we got a partial prove of applicability of the framework to complex processes and its usefulness.

References

1. Bider, I. and Perjons E. Preparing for the Era of Cloud Computing: Towards a Framework for Selecting Business Process Support Services. *LNBIP*, No 113, 2012, Springer, pp. 16-30
2. *Standard for Integration Definition for Function Modeling* (*IDEF0*), National Institute of Standards and Technology (NIST), FIPS publication, 183, December 1993.