Proceedings of the 5th WSEAS Int. Conf. on Software Engineering, Parallel and Distributed Systems, Madrid, Spain, February 15-17, 2006 (pp8-13)

Change Management Process on Database Level within RUP Framework

ZELJKA CAR*, PETRA SVOBODA**, CORNELIA KRUSLIN**
*Department of Telecommunications
Faculty of Electrical Engineering and Computing, University of Zagreb
Unska 3, 10000 Zagreb
CROATIA
www.tel.fer.hr/zcar

**Research and Development Department
KATE-KOM d.o.o.
Drvinje 109, 10110 Zagreb
CROATIA
www.kate-kom.com

Abstract: - The paper presents a tailored RUP subprocess model with respect to database design and implementation. Paper deals with documentation development for database team working concurrently on the database implementation. Paper focuses on RUP construction phase of application life cycle, with the emphasis on Implementation and Change Management discipline on the database level. Change management process model is proposed for database development, which is compliant to tailored RUP framework. Integration points are RUP defined artifacts, roles, and activities. Paper presents new artifacts added to tailored RUP model, as well as all other issues and their solutions for efficient organization of database development team.

Key-Words: RUP, tailoring RUP, change management process, database design, implementation

1 Introduction

One of the basic for a software project to be successful is that an appropriate underlying software process is defined in such a way to respond to the project's needs. *Rational Unified Process* (RUP) is a software development process applicable to a variety of projects and able to accommodate varying project needs. RUP is an iterative process, divided into four main stages [1]: *Inception*, *Elaboration*, *Construction*, and *Transition*. Throughout the course of the project, each of these stages requires different disciplines to be applied, at varying amounts of workload. Generally, RUP is organized into nine disciplines: *Business Modeling*, *Requirements*, *Analysis and Design*, *Implementation*, *Test*, *Deployment*, *Configuration* and *Change Management*, and *Project Management* and *Environment*. The concepts of roles and activities answer the questions of "*who* is doing which task", "*what*" is the task, and "how" to accomplish the task. Artifacts are results of performing the activities. For each artifact within the RUP there is a template available, sometimes accompanied with the example artifact [1]. However, the list of roles, artifacts, and activities, by itself, does not constitute a process. These process elements and their interactions have to be organized in a proper way. This is done with workflows. Following this methodology ensures production of high quality software [1].

RUP is offered as a framework so it presents many extension points and tailoring opportunities. When adapting RUP [2] to fit a specific project, it is important to keep the integrity of RUP as a framework. A tailored RUP model still defines a project in terms of phases and set of disciplines. However, some disciplines, artifacts and roles may be omitted or added. Existing literature and work does not provide with many guidelines on how to tailor RUP in this way, though a clear need for such guidelines exists [2].

This paper presents a tailored RUP subprocess model with respect to database design and development. RUP was originally tailored for the case study of information intensive services development, in the scope of a research project between the KATE-KOM company and the University of Zagreb, Faculty of Electrical Engineering and Computing. The applications deal with notification services via SMS messages, enabling a fast, simple, and cheap system capable of exchanging information among different customers and application providers. The complete approach used in tailoring RUP for the development of the start-up application is

described in [3]. The development of the subsequent applications within information-intensive services set was used for fine-tuning the original tailored RUP model.

Although a complete RUP tailored model covers the entire software lifecycle, *Construction* phase and database development activities have been chosen for this paper as the most illustrative. The development environment is presented and a case study project is defined in terms of size, complexity, and length to allow a better understanding of the utilized approach for defining a database development process. We specify all the problems and issues regarding the database development during the project, and a solution is proposed in the form of a database development process model. The proposed process model is fully compliant and joined to the tailored RUP process model through the requested RUP database oriented artifacts, defined at the project level.

The paper is organized as follows: First section shortly indicates tailoring guidelines for the project case study. Favorable tailoring decisions are formalized and stored as a part of the process model, and thus can be applied in future projects. In the second section database design and implementation process model is described. Finally database development process model for RUP construction phase is introduced as well as issues and solution for creating documentation for change management at database level.

2 Description of the tailored RUP model

First we will mention the basic principles of the tailoring approach. It is fully described in [3]. A software project case study is considered as a small project with a short product cycle (a small team with less than fifteen members, project duration of less than one year). The number of documents in such projects tends to be smaller and less detailed. Other characteristics of a small project is frequent adding of new people to the project team, and focusing project efforts on knowledge management and efficient knowledge transfer without significant time penalties.

The baseline for RUP tailoring was a selection of only those key development activities indispensable for delivering a high quality software product [3]. Tailored RUP features are as follows [3]:

- *Identifying and prioritizing project risks.* Process artifacts aiming in risk mitigation are more detailed, and their structure as agreed within the project. Each conclusion, agreement, or solution to a problem occurring during the development process must be formalized within the most appropriate artifact for a particular development discipline.
- Concurrency in development activities is highly encouraged to attain a more agile development.
- An effective software configuration management process is established within the whole application life-cycle.

Configuration management (CM) is concerned with managing evolving software requirements within different lifecycle phases. Since nowadays, a software development team faces ever-changing requirements as the result of hectic market needs, effective project level CM should control the costs, deadlines, resources, and efforts involved in making changes to the application. The project CM board consists of those development team members who carry out key development roles and are experienced enough to reasonably analyze the impact of a particular change on the whole project. Within the tailored RUP model in the project case study, the proposed CM is defined both on project and database level, for controlling the implementation of the changes .

In the case study, a database was created, implemented, and tested on MS SQL Server 2000. The database contains a great amount of data about users, functionalities and charging details regarding information intensive service. Other database technologies and CASE tools used in the project case study are shown in Table 1. It is broken down by the tool's product name, and the reason why the tool was chosen for a particular aspect of the case study.

Table 1: Tools used in our case study development

Tools	Purpose	Description
Rational Requisite Pro	Input of requests, creation of vision, creation of dictionary, use case description, etc.	Request are systematically documented and all developers with granted permission supplement and change requests.
Rational Rose	Database Modeling	UML diagram creation, system functionality descriptions, database functionality description, database objects description, etc.
Rational Soda For Word	Documents generation	
MS SQL Server – Enterprise Manager	Implementation	Implementation of database objects: tables, stored procedures, user defined functions, jobs, etc.
MS SQL Server – Query Analyzer	Implementation and testing	Database general scripts, database test scripts, etc.
MS SQL Server – Profiler	Testing	Observing of database behavior through the work.
Text processing tools (ex. MS Word)	Document generation	Generation of documents made for corporation between developers.

3 Database development process model for RUP construction phase

After the initial *Requirements* discipline activities finish, these basic artifacts are created: *Requirements Management Plan, Software Requirements Specification, Supplementary Specifications and Use-Case Model*.

These artifacts are foundation for:

- Determining general data and their attributes to be contained in a database;
- Defining relations between different kinds of general data;
- Representing general data and associated attributes in an object-oriented manner by using a design model containing use-case realizations, design package, and design class diagrams.

Quality database design in this phase is crucial for proper input of changes to the database in later project phases. In our case study, requirements were changing rapidly and new ones were added very frequently. If database design was inadequately developed, errors would come up soon with new the addition of new requirements. Database design errors are the hardest to fix and the most expensive due to the possibility of damage to the entire database later on in the project. A valid database design should be open for introducing future change requests. Following guidelines in [5], we have developed design model in the case study containing use-case realizations, design package, and design class diagrams.

Figure 1 shows *design class diagram* as a part of *design model* in the project case study. Application contains one database with data users, services, charging information etc. and these data are object-oriented and represented in the class diagram. According to these classes new tables are created in database.

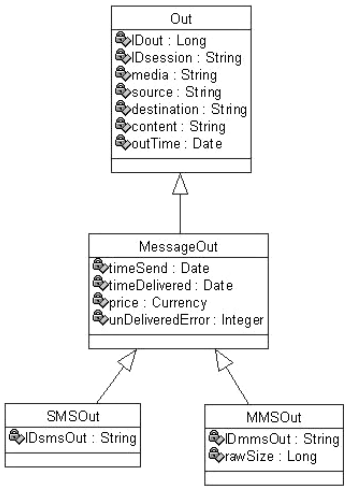

Figure 1: Example of design class diagram from design model in the project case study

On the basis of the *design model* and requirements oriented artifacts mentioned before, *data model* creation is started. In *data model*, objects and their attributes are created relationally, according to the conversion between *database design* and *data model* proposed by RUP. By this conversion *classes* in the *design model* become *tables* in the *data model*, *attributes* in the *design model* become *database columns* in *data model* etc. *Database data model* is created and has clearly specified database structure. Well defined and validated *data model* ensures avoiding conflict situations that may come up during database development and changing.

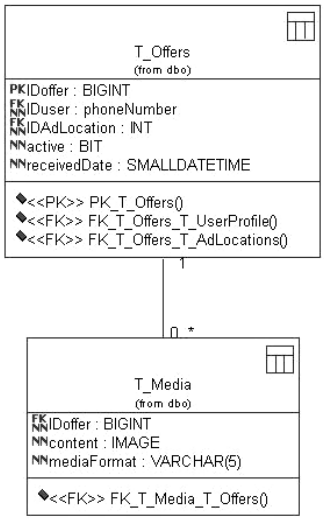

Figure 2: Example of data model in the project case study

Figure 2 shows part of *data model* created in the project case study. In the shown model data are represented as *tables*, *attributes* and *constraints*, and show the true database condition. *Data model* may contain stored *procedures*, *triggers*, *user-defined data*, *functions* etc.

Within given tailored RUP model for the project case study, database development team has associated two roles. First role is *database developer*. This role is assigned to one or more people. Second role is *database development leader*. This role is assigned to only one person. In the case study *project database development team* (DBD team in the following text) consists of one *database development leader* (DBD leader) and three *database developers (DB developers)*. Database development team activities and responsibilities are organized as follows:

- DBD *leader* participate as a DBD *team representative* in project CM board and is engaged in decision making process regarding requested changes. All decisions from CM board are documented within RUP artifact *Change Requests Decisions* document, created by using the tool *Rational RequisitePro* (Table 1).

- The way how change is implemented in the database is discussed in the DBD meetings with all DBD *team members* included regularly.
- DB is divided into several *logical units* (LU) covering different application business logic features. Each developer is responsible for design and implementation of particular DB LU.
- All DB *developers* should have the understanding of the overall database functioning, purpose and assignment, but they are not ought to be familiar with the implementation details for those DB LU they are not responsible for.
- *Database design model* and *data model* are the results of the collaboration of all DBD *team members*. DBD *leader* is responsible for validating the models and managing implementation and testing activities.
- Each DB *developer* has a local copy of a database and on the server there is a master database.
- Each DB *developer* is responsible for implementing assigned DB LU, and for testing it on the local database copy. DB *developers* create stored procedures, jobs etc. in MS *Query Analyzer* and MS *Enterprise Manager* (Table 1).
- Each DB *developer* is responsible for documenting implementation and testing of assigned DB LU.
- In the defined time period all DB local implementations are imported on the central DB for which DBD *leader* is responsible for. The role of project tester is responsible for testing integration of DB and other application modules.
- One DBD *team member* is appointed for writing basic database artifacts defined by RUP, design class, use case models and to develop internal database documentation: DBD *Change Input*, DBD *Change Control Document, Implementation and testing records*, DBD *Testing records*, DBD *Proposals and recommendations*.

3.1 Change management process on database level

There are some situations when changes must be entered into a database, sometimes unfortunately even right before software product delivery. The reason for this is efficient response to market needs. But even minor database change entry can cause chain reaction and finally crash the database. Database change just before software product delivery is called high-risk situation, and has to be under full control. Different database implementations can have a same interface to other non-database modules. Introducing inadequate change in a database can have negative impact on the functioning on the whole application.

Assumptions inherited from the project CM process are:

- All introduced changes must be documented. This should prevent conflict between different application modules. Process should ensure that all DB *developers* are aware of all database changes.
- Changes are classified into *major* and *minor*. Classification is done by DBD *leader* who analyze change request approved by project CM board. *Minor changes* on database level in our case study are changes affecting less than 5 objects. This type of changes does not implicate significant reconstruction of stored procedures, jobs, tables etc. All others are considered to be *major changes*. Change request is documented in artifact DBD *Change Input*. Knowledge about major changes should be shared among all team members and should be documented in DBD *Change Input*, DBD *Change Control Document.*
- If more than one database *developer* make changes in database at the same time, database testing and error handling is disabled. Database has to be unchanged in defined time period. Only changes that will appear in defined time period are documented.

If more than one database developer is involved in same project database creation, establishing a *change input period* is needed. *Change input period* is a clearly defined period for entering changes into database and documenting it. In our case study a change input period had a value of one week. Each database *developer* has a local version of the main database and enters changes through defined change input period into it. At the end of the change input period database *developers* join their changes to the main database version. Exactly at specified day a DB *developers* enter tested changes they implemented on their local database copy into central database on the server. New DB objects are imported to existing central database using tools like MS *Enterprise Manager* for database objects import (Table 1) or in case of minor changes DB *developers* enter changes into central database manually. DBD *leader* supervises integration of all local changes to the main database. If integration problems happened, DBD *leader* is responsible for coordinating DB *developers* to solve a specific problem.

DB *developers* tasks in CM process at database level are:

- Taking care about assigned minor change independently and implementing it, test and document by itself. This should be done during defined change input period.
- Participating in the modification of database design model, data model and other artifacts impacted by major change.

- Reviewing each proposal of change implementation proposed by other DB developers in DBD *Change Control Document*.
- DB *developers* have access to shared document repository containing different database objects constructions. They use predefined templates for documenting the change. These documentation templates are fulfilled with advises, instructions and examples arising from previous projects, and they present effective way of reminding about different important issues that could be eventually omitted.

DBD *leader* tasks in database CM process are:

- Assigning changes to the DB *developers*.
- Consulting DB *developers* through process of new database object creation in design model, data model, implementation of complex solutions, etc.
- Reviewing DB *developers'* major change propositions.
- Organizing and leading database meetings. All members of DB *team* should attend database meetings regularly.

Figure 3 shows proposed CM process at database level Process input is change request arrival (1). The change has to be accepted by project CM board. Database CM process is described as follows:

(2) DBD *leader* receives documented change request in *Change Requests Decisions* document, containing database change specified as a new requirement.

(3) DBD *leader* classifies change as a major or minor.

(4) DBD *leader* assign change to the particular DB *developer*. In case of minor change, change is assigned to only one *developer*. In case of major change, change is assigned to two or more *developers*.

(5) Before implementing assigned change, DB *developer* reads appropriate documentation with examples and existing solutions stored in DB *team* document repository in order to find the best solution for the change they have to implement.

(6) DB *developer* proposes his solutions for assigned change request and writes down his proposal in the DBD *Change Control Document* (Table 2).

(7) DBD *leader* and DB *developers* write down comments to proposed solutions in the same document.

(8) In case when proposed solution is inappropriate database development team returns to state (6). Otherwise it will be accepted and implemented in further steps continuing into state (9).

(9, 10) DB *developers* implement and test accepted change solutions.

(11) DB *developers* document testing. Documentation consists of test inputs, expected outputs and test results for every part of the code he has done. Also there is indicated the test script is created.

(12) If database change implementation is not successfully accomplished, DBD team returns to beginning of database change implementation (state 9). If database change implementation is finished implemented database change is integrated with the main database (13).

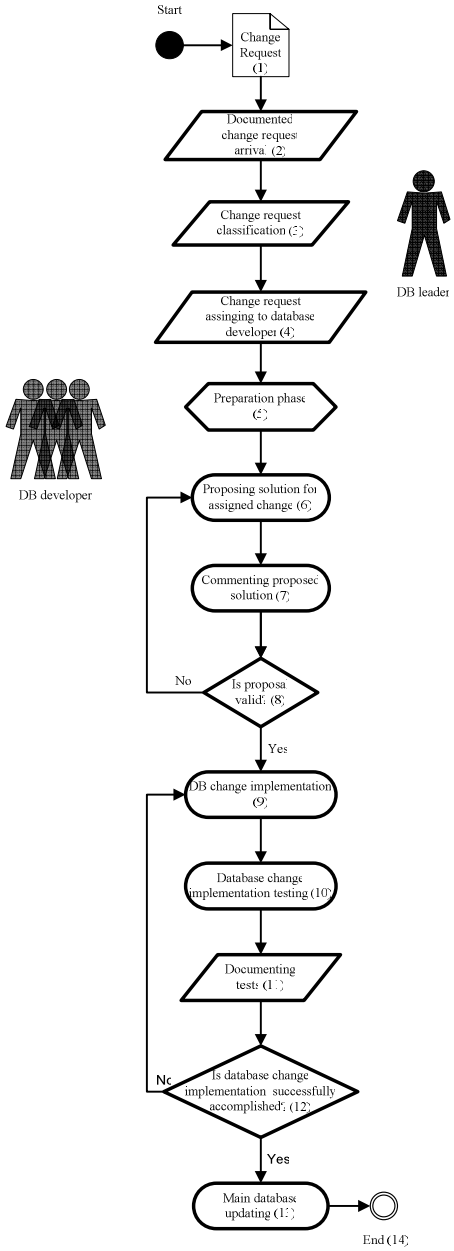

Figure 3: CM process at database level

3.2 Creating documentation for change management at database level

Complete documentation for change management at database level is indispensable for knowledge sharing and knowledge preserving between DBD team members. This kind of documentation enables communication between database developers, their

leader and provides better insight in database development activities. Documents for CM at database level are origin RUP artifacts and added artifacts for covering DBD team activities and database knowledge management and transfer to new DBD team members.

Table 2 shows documents for database design defined by RUP.

Table 2: Database artifact

Artifacts	Database action / RUP phase	Description	Defined by RUP
Design Model	Design / All phases	Object model describing the realization of use cases, and serves as an abstraction of the implementation model and its source code [4].	Required
Data Model	Design / All phases	Describes the logical and physical representations of persistent data used by the application [4].	Optional
Project Specific Guidelines	Design / All phases	Guidance on how to perform a certain activity or a set of activities in the context of the project [4].	Optional
DBD Change Input	Change input / Construction and Transition	Detailed description of every database object, including information for database team.	Not defined
DBD Change Control Document	Change input / Construction and Transition	Description of proposed solution, other DB developers comments.	Not defined
Implementation and testing records.	Change input / Construction and Transition	Description of database implementation process, testing process and record of possible problems in those processes.	Not defined
DBD Testing records	Testing / Construction	Testing process description including test scripts description.	Not defined
DBD Proposals and recommendations	Testing / Construction	Supplementing the existing and creation of new proposals and recommendations.	Not defined

Table 2 also contains documents used in our case study that are not defined by RUP:

- DBD *Change Input*. Detailed description of every new database object, including information who did it, what is it for, who instructed developer, when object was created, and was it successfully accepted by other DB developers code in the main database. RUP defines similar documentation through reverse engineering where database object has detailed description, but it does not contain enough information for use in forward engineering. A document preserves information about DBD team member's responsibilities and efforts.
- DBD *Change Control Document*. Preserves information about change implementation proposals and objections, and is helpful for maintaining open database structure ;
- *Implementation and testing records* and *DBD Testing records*. Preserve information about entered changes that are important for further database testing, integration and maintenance of the overall application.

DBD leader is responsible for providing all necessary documentation, templates and possible best practice from previous projects and for monitoring the process of database documentation management.

4 Conclusion

The paper presents a tailored RUP subprocess model with respect to database design, implementation and change management. RUP was originally tailored for the case study of information intensive services development, but experience of start-up application showed need for better formal modeling of database development process, especially regarding change management. Therefore the process of change management on the database level was formalized and supplemented with additional artifacts in order to document and transfer database knowledge and efforts. From our experience, this approach was helpful and beneficial for the overall project, even after the application delivery for both perfective and corrective maintenance process, especially for better knowledge transfer through the project.

References:

[1] P. Kruchten, *The Rational Unified Process: An Introduction*, 2nd Edition, Addison Wesley Longman, 2000.

[2] G. K. Hanssen, H. Westerheim, F. O. Bjørnson, Tailoring RUP to a defined project type: A case study. *Proc. 6th International Conference on Product Focused Software Process Improvement* (PROFES'2005), Oulu, Finland, 2005, pp. 314-327.

[3] Ž. Car, O. Labor, A. Carić, D. Huljenić, Tailoring RUP: E-School Project Case Study. *Proceedings of the Conference Telecommunications & Information MIPRO 2004*, Rijeka, Croatia, 2004, pp. 27-32.

[4] M. Hirsch, Making RUP Agile. *Conference on Object Oriented Programming Systems Languages and Applications archive - Practitioners Reports.* Seattle, Washington, 2002.

[5] E.J. Naiburg, R.A. Maksimchuk, *UML For Database Design*, Addison-Wesley, 2001.