Nicolas Guelfi (Ed.)

LNCS 3475

Rapid Integration of Software Engineering Techniques

First International Workshop, RISE 2004
Luxembourg-Kirchberg, Luxembourg, November 2004
Revised Selected Papers

Springer

Nicolas Guelfi (Ed.)

Rapid Integration of Software Engineering Techniques

First International Workshop, RISE 2004
Luxembourg-Kirchberg, Luxembourg, November 26, 2004
Revised Selected Papers

Springer

Volume Editor

Nicolas Guelfi
University of Luxembourg
Faculty of Science, Technology and Communication
1359 Luxembourg, Luxembourg
E-mail: nicolas.guelfi@uni.lu

Library of Congress Control Number: 2005925572

CR Subject Classification (1998): D.2, F.3, K.6.1, K.6.3

ISSN 0302-9743
ISBN-10 3-540-25812-4 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-25812-4 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11423331 06/3142 5 4 3 2 1 0

Preface

RISE 2004 was an international forum for researchers and practitioners interested in integrated and practical software engineering approaches that are part of a methodological framework and which apply to both new and evolving applications, technologies and systems. The ERCIM (European Research Consortium for Informatics and Mathematics) RISE working group selected application areas such as the Web, mobility, high availability approaches, embedded approaches and user interfaces in specific industry sectors comprising finance, telecommunications, transportation (avionics, the automotive industry) and e-government. Considered research issues in these areas pertain to the following software engineering domains:

- software/system architectures
- reuse
- testing
- model transformation/model-driven engineering
- requirements engineering
- lightweight formal methods
- ASE tools

All papers submitted to this workshop were reviewed by at least two members of the International Program Committee. Acceptance was based primarily on originality and contribution. We selected for these proceedings 12 papers amongst 28 submitted, and an invited paper.

The organization of such a workshop represents an important amount of work. We would like to acknowledge all the Program Committee members, all the additional referees, all the Organization Committee members, the University of Luxembourg, Faculty of Science, Technology and Communication administrative, scientific and technical staff, and the Henri Tudor Public Research Center.

RISE 2004 was mainly the supported by ERCIM, the European Research Consortium for Informatics and Mathematics, the "Ministère de l'enseignement supérieur et de la recherche," and by the "Fonds National pour la Recherche au Luxembourg."

November 2004 Nicolas Guelfi

Organization

RISE 2004 was organized by the University of Luxembourg, Faculty of Science, Technology and Communication.

Program Chair

Guelfi, Nicolas	University of Luxembourg, Luxembourg

International Program Committee

Arve Aagesen, Finn	NTNU, Norway
Bertolino, Antonia	CNR-ISTI, Italy
Bicarregui, Juan	CCLRC, UK
Bolognesi, Tommaso	CNR-ISTI, Italy
Born, Marc	Fraunhofer FOKUS, Germany
Buchs, Didier	SARIT, University of Geneva, Switzerland
Dony, Christophe	LIRMM, University of Montpellier, France
Guelfi, Nicolas	FNR, University of Luxembourg, Luxembourg
Haajanen, Jyrki	VTT, Finland
Issarny, Valérie	INRIA, France
Klint, Paul	CWI, The Netherlands
Moeller, Eckhard	Fraunhofer FOKUS, Germany
Monostori, Laszlo	SZTAKI, Hungary
Pimentel, Ernesto	SpaRCIM, University of Malaga, Spain
Romanovsky, Sacha	DCS, University of Newcastle, UK
Rosener, Vincent	FNR, Henri Tudor Public Research Center, Luxembourg
Savidis, Anthony	FORTH, Greece
Schieferdecker, Ina	Fraunhofer FOKUS, Germany

Organizing Committee

Amza, Catalin	University of Luxembourg/DISI, Genoa, Italy
Berlizev, Andrey	University of Luxembourg, Luxembourg
Capozucca, Alfredo	University of Luxembourg, Luxembourg
Guelfi, Nicolas	University of Luxembourg, Luxembourg
Mammar, Amel	University of Luxembourg, Luxembourg
Perrouin, Gilles	University of Luxembourg, Luxembourg

Pruski, Cédric	University of Luxembourg, Luxembourg
Reggio, Gianna	DISI, Genoa, Italy
Ries, Angela	University of Luxembourg, Luxembourg
Ries, Benoît	University of Luxembourg, Luxembourg
Sterges, Paul	University of Luxembourg, Luxembourg

Sponsoring Institutions

This workshop was supported by ERCIM, the "Ministère de l'enseignement supérieur et de la recherche", and by the "Fonds National pour la Recherche au Luxembourg."

Table of Contents

Integration of Software Engineering Techniques Through the Use of Architecture, Process, and People Management: An Experience Report

Christopher Nelson[1] and Jung Soo Kim[2]

[1] Siemens Corporate Research, Princeton NJ 08540, USA
Christopher.Nelson@siemens.com
[2] Carnegie Mellon University, Pittsburgh PA 15213, USA
jungsoo@cmu.edu

Abstract. This paper reports on the experiences of integrating several Software Engineering techniques by the Rapid Prototyping group of Siemens Corporate Research. This experience was gained during our recent project that involved developing a web-based, workflow-driven information system. The techniques integrated for this project included agile and iterative processes, user centered design, requirements discovery and maturation, and test-driven development. These techniques were integrated and supported by a proprietary process entitled "Siemens Rapid Prototyping" (S-RaP), a software architecture, and project management techniques. This paper will detail the specific characteristics of S-RaP, the software architecture, and the project management techniques that supported the integration of the above listed software engineering techniques. We will also report on our experience with their effectiveness and our thoughts on future enhancements in all three areas.

1 Introduction and Background

Siemens Corporate Research, Inc. (SCR) is the Research and Development organization of Siemens USA. Over the past several years, the Siemens Rapid Prototyping group at SCR has been developing, formalizing, and maturing an agile and iterative process entitled S-RaP. This process was born from the necessity to manage a series of projects that were emphasizing usability, as well as requiring the maturation of requirements throughout the development process.

To support projects with these characteristics, a process that incorporated User Centered Design (UCD) was fundamental for success. Unacceptable to our customers was a heavyweight requirements engineering phase, or for the addition of UCD practices to lengthen the time-to-market. In addition to requirements maturation and UCD practices, we also needed to include standard Software Engineering practices such as software architecture, design, and testing.

The rest of this paper discusses characteristics of the S-RaP process, architectural and design decisions, and characteristics of people management that we, at SCR, have utilized to combine multiple Software Engineering techniques. This combination has allowed us to include requirements maturation and UCD practices into our processes without lengthening time-to-market for the prototypes and products we develop.

2 S-RaP Process

The S-RaP process was designed and developed to aid in the execution of projects such as the one described in this paper. Key characteristics of these projects include a very short timeframe often in the order of 3 to 8 weeks from initial requirement capture to final delivery, vague and very high-level requirements that need to be explored and matured, and a heavy emphasis on usability.

The key aspects of the S-RaP process that allow it to successfully govern such projects, as illustrated in Figure 1, are that it is iterative, incorporates User Centered Design techniques, uses a single artifact throughout the life of the project that facilitates communication between the different process threads [9, 10], and executes three parallel threads of Requirements Engineering (RE), User Interface Design (UID), and Software Build (SB) [11].

2.1 Parallel Execution of Process Threads

By executing the three parallel threads of Requirements Engineering, User Interface Design, and Software Build in our process, we are able to integrate these three software engineering techniques in a unique way to help make our projects successful. Our process enables us to capture, over time, accurate requirements by allowing the requirements to mature throughout the development phase, as well as incorporating UCD techniques into our software development process. We have been able to do so without increasing our expected time-to-market.

The parallel execution of these activities is notably different from the serial execution of these activities, specifically in the coordination of, and communication among, the three threads in the process. With parallel execution, there is a need for a well-defined channel or mechanism of communication between all the groups involved. This is the case because parallel execution introduces a high risk of duplication of effort, or wasted effort from work done by one group in a direction inconsistent with the rest of the groups.

Our process incorporates the use of a single software specification artifact throughout the life of the project [9], as well as the functional prototype as a boundary object to facilitate communication between the groups functioning in parallel [10]. Also used to help mitigate these risks were short iterations, the co-location of all our team members and teams, and aspects of the architecture discussed later.

2.2 Short Iterations

All phases and threads in our process are executed in an iterative manner with very short iterations that are approximately one week in length. This serves several purposes. First, it ensures constant communication between all the groups

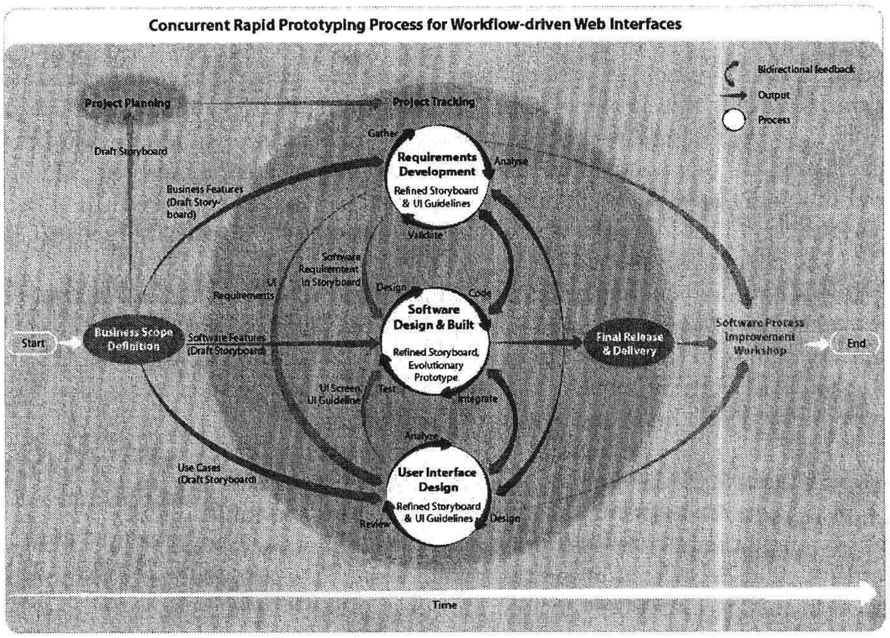
Concurrent Rapid Prototyping Process for Workflow-driven Web Interfaces
Bidirectional feedback
Output
Process
Project Planning
Project Tracking
Draft Storyboard
Business Features (Draft Storyboard)
Gather
Requirements Development
Refined Storyboard & UI Guidelines
Analyse
Validate
Software Requirement in Storyboard
UI Requirements
Design
Code
Start
Business Scope Definition
Software Features (Draft Storyboard)
Software Design & Built
Refined Storyboard, Evolutionary Prototype
Final Release & Delivery
Software Process Improvement Workshop
End
UI Screen UI Guideline
Test
Integrate
Analyze
Use Cases (Draft Storyboard)
User Interface Design
Refined Storyboard & UI Guidelines
Review
Design
Time

Fig. 1. Siemens Rapid Prototyping (S-RaP) Process Model

since inputs to one group for the start of an iteration are generated as outputs from the iterations of the other groups [11]. This is important to keep all groups in synch and to keep the team as a whole moving in a consistent direction as directed by our customers. Second, it allows each group to mature their aspect of the project based on feedback from the other groups. The RE team can mature the requirements by presenting usability feedback and the latest version of the application to the customer in design review meetings. The UID team can use the most recent version of the application as well as the newest version of the requirements to mature their UI definition and style guide, as well as continue usability testing. The SB group can continue to evolve the application's architecture, design, implementation, and tests based on matured requirements and UI feedback. Third, it allows for continuous usability testing, acceptance testing, unit testing, and market validation. This is important because of the dynamic nature of the projects. It helps to catch and fix issues as early as possible and acts as a safety net to protect against accidentally dropped tasks. This is very important due to the increasing cost of change as a project progresses [2, 3, 4].

2.3 Co-located Multi-disciplinary Teams

In our project we tried to maintain the co-location of our team members, and our teams. Our RE team was slightly distributed in the sense that two to three of the members were from our customer's organization which is located in a different state, and thus were not on site with the rest of the RE team. It is in this team that we notice the most difficulty in maintaining constant and fluent communication. For this reason we assign a single point of contact from our organization to our customer who is responsible for maintaining the flow of communication.

This seems to work well because it eases the burden of communication from our customer's perspective by eliminating redundant communication. The high level of communication stemming from co-located teams simplified the integration of everything from the use of an agile process, to parallel threads of RE, UID, and SB, to continuous testing.

In addition to facilitating communication, the co-location of multi-disciplinary teams also made it easy to put together sub teams of individuals with complementary skill sets. This was especially beneficial when we had difficult problems to solve that required expertise in several different areas such as Human Computer Interaction (HCI) and Software Engineering.

In such situations we found or noticed two methods useful, one for predefined complex tasks, and the second for emergent complex tasks. With the predefined complex tasks with specific deadlines, defining formal multi-disciplinary teams and assigning them to the task worked very well. In other cases, difficult problems would just emerge from other tasks and would not have concrete completion dates. In this case we often noticed impromptu hall meetings between members of the different teams. This was made possible by the co-location of the teams. In both cases, the co-location of teams really helped in integrating the UID team and UCD practices with the rest of the core team and practices.

3 System Architecture

The given business context of the project imposed a few significant risks on us that had to be dealt with on system architecture level. Multiple releases were to be made in sequence, and each release was tied to a very short timeframe with a strict deadline. Thus, we required parallel development of requirements and no-overhead integration. The customer did not have specific and concrete requirements from the beginning and usually asked us to demonstrate tangible and executable artifacts to get their ideas particularized. Moreover, the requirements changed frequently throughout the development. Therefore, we desired the ability to easily modify the existing implementation, at a given point in time, without breaking down the other existing parts. Yet, the user interface should be highly usable and consistent across different workflows and releases. Obviously development-from-scratch would not work at all under such circumstances, and a more structured and systemized design method was necessary. [6, 7, 8]

At the beginning we started with a very simple architecture of a client-server style and used MVC pattern [5] in the server. Throughout the development, however, we found the need for a better architecture that would facilitate dealing with the business constraints, and as a consequence, the architecture evolved. In this section we will explain the key aspects of the current architecture and its evolution throughout the development process.

3.1 Three-Tiered Style

Our system architecture is a typical example of a three-tiered style with a client tier, a server tier, and a database tier. The client tier is the simplest and thinnest tier of the system, consisting of a single standard web browser that can send HTTP requests and render returned HTML pages on the screen. The database tier is responsible for storing operational data, as well as providing a mechanism for read and write access for the data. The server tier sits in between the client tier and database tier. It recognizes requests from the client tier, fetches data from the database tier if needed, and provides results back to the client tier. Each tier is allowed to interact only with the other adjacent tiers through the specified communication protocol as illustrated in the Figure 2.

When the project was started there was no database tier, and the system consisted of a very thin client tier and a monolithic server tier. Throughout the development there were a lot of requirement changes, most of which are related to the UI requirements. We factored out the part of system that were not affected by such UI requirement changes and made it into a separated database tier.

This architectural evolution of logical separation facilitated application of several SE techniques to the project. Incorporating requirement changes was made easier because only the server tier was to be modified, leaving the database tier untouched or just adding new data objects without modifying the data access protocol. Macro-level testing was done more efficiently because each tier could be tested separately and independently. Especially the testing of database tier was conducted using the data access protocol without the server tier, and this eliminated a big source of defects, which reduced the time to locate and fix a defect dramatically. Also parallel development of a new workflow was done more systematically because UI parts and non-UI parts could be developed concurrently and integrated using the data access protocol.

For the design of the data model, we employed the actual object model from an existing Siemens product. Though it was an over-specified design, due to a few aspects of the data model that were not used in our development, we found this was a wise move because the overhead was from a product from which we

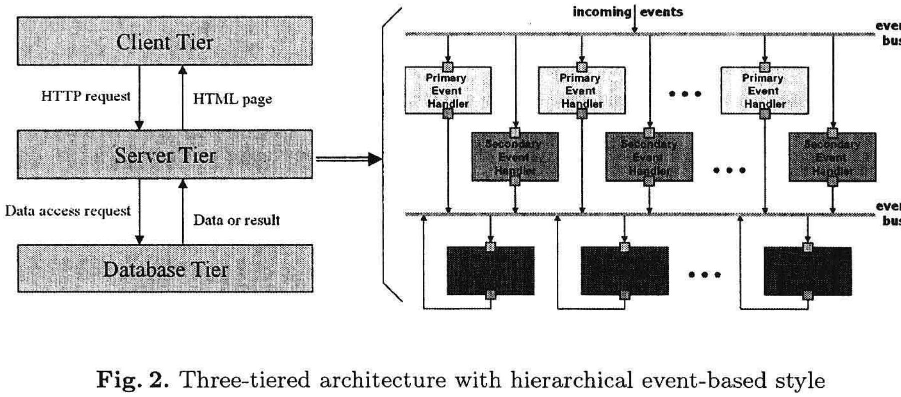

Fig. 2. Three-tiered architecture with hierarchical event-based style

could derive future requirements of our system. There actually were a few cases where we received new requirements from our customer, and the data support was already implemented in the database tier.

3.2 Event-Based Style

The internal architecture of the server tier is an instance of event-based style, which was a very natural decision because its job is to process asynchronous requests from the client tier. In other words, its execution is triggered only by the requests from the client tier, which are to be interpreted into events and processed in the server tier. Internally the server tier is structured as a collection of small event handlers, and each is responsible for handling specific events in specific contexts.

What distinguishes our architecture from general event-based style with multiple event handlers is that the relation between event handlers is hierarchical. Some event handlers have more privileges and responsibilities, but also some event handlers receive events directly from the client tier and selectively forward events to the other event handlers in the lower level of the hierarchy. Of the event handlers in the top level of the hierarchy, only one event handler is made active at any given point in time. The active event handler always receives incoming events, and is responsible for activating its replacement event handler when necessary.

When the project was started there was no hierarchy between event handlers. This was because each event handler was responsible for an independent portion of the requirements. This enabled easy allocation of requirements to developers. Throughout the development we found that there was a lot of redundancy in the event handlers that were implementing similar requirements. Such redundancy not only wasted our development resources, but also caused confusion and conflicts. The continuously changing requirements made maintaining consistency between redundant implementations a source of pain and defects. For the rapid prototyping we needed to eliminate that redundancy, and as a result we factored out common functionality from similar requirements into separate reusable event handlers that could be used by the originating event handlers. As a result the architecture was evolved accordingly to incorporate such structural changes in the design level.

For this reuse of event handlers we designed a unified event forwarding mechanism so that plugging in a new event does not cause any structural inconsistency or side-effects. Then, the implementation of an event handler simply involved filling in a template that already had the event forwarding mechanism built-in with the new functional behavior. Moreover, the internal details of an event handler were completely independent of the hierarchy of event handlers because event handlers have no way of telling where the events come from. The structure of the hierarchy was only important in the integration of event handlers.

This architectural evolution gave us a new edge in business. As stated, elimination of redundancy reduced the time for delivery. This gain has accumulated as