Reengineering Legacy Application to E-Business with Modified Rational Unified Process

G. Jeyaraman, Kumar Krishnamurthy and V. V. S. Raveendra

Abstract
Experience in reengineering a legacy application into a web based J2EE system with modified Rational Unified Process (RUP) is presented. RUP is adopted into an onsite-offshore development model along with ISO 9001 and SEI CMM Level 5 standards. The new application has above 2500 code components and the effort is about 100 person years. For the benefit of software development community, some of our experiences in design, development, testing and project management are elaborated as generalized concepts. We have demonstrated that development process could be improved with lessons learnt from the initial iterations. The three views of a web application are explained and the translations between the layers are discussed. Benefit of continuous integration is highlighted. Various types of dependencies to be taken into account for sequencing the development are elaborated. The levels of testing in iterative development are mentioned. The importance of adaptive team structure and various parameters guiding iteration planning are dealt with. A simple estimation model based on types of transactions is presented. Finally, a fine grained risk management concept that can integrate with the development process is proposed.

1. Introduction
The paper deals with our experience in the execution of a large J2EE (Java 2 Enterprise Edition) project. Business domain of the application is Unemployment Insurance in the USA. The customer had a legacy application on mainframe developed long ago in Cobol with VSAM (Virtual Storage Access Method) and IDMS (Integrated Database Management System). The aim of the project is to reengineer the legacy application into a state-of-the-art web based system with J2EE as the technology and Rational Unified Process (RUP) as methodology. RUP is an iterative life cycle development model from Rational Inc. More interestingly the RUP was to be implemented in onsite-offshore model satisfying ISO 9001 (International Standards Organization) and SEI CMM Level 5 (Software Engineering Institute, Capability Maturity Model) quality processes. Before embarking on to details, we survey papers relevant to e-business development.

The lessons learnt by Dennis in [1] are mostly user interface related. His experience is from the early days of e-business application development and is not discussed in relation to iterative development or J2EE. Pooley et al. [2] discuss defect data collection and analysis in web application development. Iterative development model perspective is not provided. Hendrix and Schneider [3] share their experiences in developing software with spiral model. Their points are related to execution of a small/medium real time application at NASA. Sotirovski [4] discusses in detail the integration aspects in iterative development. The experience shared is related to Air Traffic Control domain and the technology aspects (like J2EE) are not dealt with. Interestingly, their [3,4] experiences are positive and conclude that spiral model is a realistic one for software development.

Various points to be addressed in general while testing e-commerce systems are presented in [5]. The affect of iterative development on testing and process to be followed with J2EE are not presented. Different parts of an application could have different quality requirement. Experiences in managing such diverse criteria are available in [6]. However, the project we have dealt with...
Ambler shares his lessons learnt in two different Internet-based development projects in [7]. Abstractions or generalizations are not mentioned, for others to make use of them. Key topics in using J2EE technology for web-based development are detailed by Altendorf et. al. [8].

Looney and Chatterjee [9] explain the business aspects to web-enable brokerage industry. Their paper does not mention about software development methodology. Gopal et.al. [10] collected data on 34 application software projects completed in onsite-offshore model during 1994 to 1996. They studied the effect of rework on elapsed time and effort. The projects were completed before web development became popular and J2EE came into being. The type of those projects (maintenance, development or enhancement) and technologies employed are not mentioned. Aversano et. al. [11] report their experience on migrating a legacy system to the web. In our case, the legacy system is quite old and it was decided to redevelop the application in J2EE instead of web enabling.

In this article, the profile of the project is mentioned briefly. Our experience in implementing RUP in onsite-offshore model with ISO and CMM standards is detailed. Some strategies implemented in the project are presented here as generic concepts that can be implemented in any J2EE project. These are related to design, development, testing and project management. Software development community can benefit greatly from these ideas.

We show that experiences in initial iterations can be quickly made use of in subsequent iterations, with iterative development methodology. There exist three views of a web application: User Interface View, Business Objects View and Data View. Software designers need to keep these views in mind. Advantage of continuous integration vis-à-vis incremental integration is discussed. Various types of code dependences to be taken into account during development and levels of testing are presented. The need for adaptive team structure is highlighted. Simple models for effort estimation and risk management are proposed.

2. Project Profile

Business domain of the application is Unemployment Insurance in the USA. The unemployed people file claims for benefits. After the adjudication of monetary and non-monetary issues (if any) claims will be processed for payments. In case the decisions on issues are not acceptable to claimants, they can appeal to either lower authority or higher authority. If claimants were employed and continued to receive benefits, overpayments can be set up against them and collected.

The application uses the following software products: Websphere Application Server 3.5, MQ Series Workflow 3.3 and Secureway Directory Server 3.1, DB2 7.1 database, JIntegra 2 and JReport Enterprise Server 4.1. The system should be accessible both from Intranet and Internet, as the user interface is browser based. The system has online interfaces with Interactive Voice Recognition application, Imaging System, Printing System, Document generation system (via JIntegra), Address Validation software (that can validate postal address), Unemployment Insurance Tax System and email server. It has batch interface with about 20 other applications. In terms of effort the project size is about 100 person years. The total number of code components (includes Java Server Pages, Servlets, EJBs etc.) is above 2500. The total duration is 15 months. The architecture of the application is n-layered (See Figure 1). Altendorf et. al. [8] mention three architectural layers. Namely, Presentation Layer, Session Layer and Domain Object Layer. Our architecture is made more granular with six layers to improve maintainability and adaptability. User Interface Layer and User Interface Control Layer, Business Control Layer and Business Layer, Data Access Layer and Data Layer are clearly separated.

Figure 1. Architecture of unemployment insurance claims system

3. RUP in Onsite-Offshore model

RUP has four phases called Inception, Elaboration, Construction and Transition. Each phase can be divided further into Iterations. More details on RUP can be found in [12].

Many enterprises outsource the reengineering, maintenance and development of applications to consulting companies. The consulting company is normally located in a different country. Many projects are executed between customer site (onsite) and consulting company site (offshore). For the project in
context, onsite location is in USA and offshore location is in India.

If the project is to adopt waterfall approach, normally the requirements would be gathered at onsite, followed by design, development and system testing at offshore. Subsequently the user acceptance testing and implementation happen at onsite. The challenge is in evolving a methodology that combines the aspects of RUP and Onsite-Offshore model meeting the ISO 9001 and SEI CMM Level 5 standards. There is not much in the current literature addressing the integrated implementation of RUP with ISO and CMM concepts. Whereas it is expected of organizations already adhering to ISO and CMM processes [13].

In the following we share RUP specific points and then quality related experience. In subsequent sections we detail the design, development and testing practices.

During iteration 1, requirements were gathered as Business Use Cases (BUCs) at onsite. BUC contains high-level business requirements as given by the users. The BUCs were further refined into System Use Cases (SUCs) at offshore. SUCs contain system specifications and are part of analysis. After review from the customer, during the design activity HTML (Hyper Text Markup Language) prototype is made first. The prototype was reviewed and approved by the users, further design, development and testing was done at offshore. We had done proof of concept for the architecture, development of infrastructure components, environment setup with application servers, configuration management between onsite and offshore (with Rational ClearCase) simultaneously in first iteration. We were successful in delivering iteration 1, but with significant delay. We had to reduce the turn around time between requirements to final delivery for iteration.

In iteration 2, representatives from users came to offshore for reducing the turn around time. Beginning from writing SUCs to system testing were done at offshore. This time the delay was reduced drastically. Finally, when all users did acceptance testing at onsite, they found that system did not meet their requirement. On doing causal analysis, we found that user representatives did not know the requirements in its entirety.

In iteration 3, a small team of analysts and designers went to onsite. They completed the SUCs and HTML prototype with application owners and users. After completion of high level design they returned to offshore and took part in development. After system testing at offshore, user acceptance testing was done at onsite. This time both ends were quite satisfied with everything done in the iteration. We did not have either schedule slippage in delivery to client or many defects in acceptance. We followed the same process for all subsequent four iterations.

Normalized schedule slippage in all iterations is shown in Figure 2. Schedule slippage is defined as \( \frac{(Actual\ Schedule - Planned\ Schedule)\times 100}{Planned\ Schedule} \). Schedule is quantified as number of days of elapsed time of iteration. Normalized schedule slippage is obtained by dividing the schedule slippage with the maximum schedule slippage. The number of components delivered in all iterations is depicted in Figure 3. It can be noticed that from iteration 4 onwards we could make bigger deliveries without schedule slippage.

![Figure 2. Normalized schedule slippage in iterations](image2)

![Figure 3. No. of components delivered in iterations](image3)

By following RUP we could identify the gaps at both ends (customer and consultant, users and developers) quickly in the early iterations. If we were to take up waterfall approach we could not have mitigated these risks until the final delivery. The experience gained in development, integration, testing, estimation and risk management in past iterations was used to plan and deliver consistent results and quality in the future iterations.

For all artifacts, members within the project team do review (Internal Quality Assurance) and for selected artifacts (20% of the total) members external to the
project team (External Quality Assurance) do the review. Artifacts are not approved for delivery to user acceptance testing unless the review effectiveness target is not met. This target effectiveness is increased for each iteration, so as to ensure that we were continuously improving. We identified common errors in the first two iterations and made checklists for defect prevention in subsequent iterations. As a result, less number of defects was reported in user acceptance testing. Implementation of quality processes, metrics collection and generation of artifacts to meet standards are done automatically through an internal Project Management tool called IPMS (Integrated Project Management System).

4. Design – A Three View Model

There exist three dimensions to any J2EE application.

4.1. User Interface View

Users who access the application via browser (a thin client) look at the application as an entity that enables them to execute certain business transactions. Therefore ‘view’ of the user is ‘transactional’.

4.2. Business Objects View

Core business logic is object oriented if designed from principles of OOAD (Object Oriented Analysis and Design). By definition, objects contain “state, attributes, and behavior”. Design and implementation of business objects in J2EE scenario involves JSP (Java Server Pages), Servlet and EJB (Enterprise Java Bean). See [14] for an interesting discussion.

4.3. Data View

The data is stored mostly in relational databases management systems like DB2, Oracle etc. For online transaction processing systems the databases are normalized. The database design would focus on data storage and access perspective.

4.4. Translations between Layers

There are two translations involved in an end-to-end (E2E) transaction. 1) The data structure on the browser side is linear or flat. Business objects have relationships like Association (one-to-many / many-to-many), Inheritance, Contains, Extends etc. The flat data structure needs to be converted into an object relationship that is often tree like (seldom graph like). 2) The objects need to be persisted in a relational database. The object relationships normally don’t map to database tables one-to-one [15].

We have addressed the above problem by designing such that Business Objects View maps one-to-one on Data View. This way one layer of translation is obviated.

5. Development Experience

In a typical J2EE application the types of components used are: Java Server Pages, Servlets, Enterprise Java Beans, Data access classes and Infrastructure classes. While 75 developers are working in parallel on various BUCs to deliver huge chunk of functionality in a short period of time, development process and integration of code are quite involved.

5.1. Integration Levels

In RUP context, integration is at various levels. Typically Business Use Cases are split into a few System Use Cases. System Use Cases are split further into a few business transactions. For a given Iteration, development activity takes place for a few Business Use Cases simultaneously. The levels of integration are explained below.

1. Transactional Level: On clicking a button on a screen, the control goes to a servlet, then to EJB (in the current project we did not use entity beans, instead we made use of pure Java classes) and then to database. After relevant database operations, on the return path control comes back to the user after display of a new screen. Let us refer to the request and response together as a transaction. Components related to a transaction need to be integrated and tested at this level. For preventing integration issues, ‘unit of work’ for a developer is coding for a transaction rather than a JSP, a Servlet or an EJB. For completing one transaction, all types of components will be touched upon.

2. System Use Case Level: A few business transactions put together become a System Use Case. All components that are related to the SUC in question have to be integrated.

3. Business Use Case Level: Components related to a BUC are integrated at this stage. Complete functionality of a BUC can be seen after this level of integration.

4. Iteration Level: All Business Use Cases developed as part of iteration have to be integrated.
5. **System Level**: Components of current iteration should seamlessly integrate with the code developed until previous iterations.

6. **Internal Interfaces Level**: At this stage application is integrated with internal interfaces like workflow server, email server, Imaging system and other applications of the same organization.

7. **External Interfaces Level**: Finally the application is integrated with internal and external interfaces in an E2E fashion.

In iteration 1, after completion of design, we assigned development of JSPs, Servlets, and EJBs to different developers. EJBs and JSPs were tested with stubs. It has added the overhead of developing stubs. We realized that too many stubs are required. Our experience in this regard is somewhat different from the Tip 7 suggested in [16]. When components were integrated for testing, we faced the integration problems. In second iteration, we assigned each transaction as a ‘unit of work’ for developers (without any stubs). From then on we did not find any problems with basic integration as such.

We employed Visual Age for Java 3.5.3 (VAJ) as a development tool. The team server of VAJ has ability to version code under development by team server administrator as well as developers connected to it. This ensured that all developers work on the correct version of code. By iteration 2 we could stabilize the process with respect to team server.

Sotirovski [4] mentions that integration is done once in iteration. This is an incremental approach. We took similar approach initially. After iterations 1 and 2, we realized that integration of code from one level to next level in the above sequence is having significant impact on the total elapsed time of development. We spent about 50% of our development time in integration alone. We found that integration of code on a continuous basis can greatly benefit in reducing the elapsed time by 40% in further iterations.

5.2. **Code Interdependencies**

In a J2EE scenario to avoid integration problems we have identified interdependencies at an early stage. The dependencies are classified into three types.

5.2.1. **Screen dependency**. On clicking of a button on screen X control goes to screen Y without any observable server side processing. Typical example is ‘next’ type of button. The two screens can be potentially from two different BUCs.

5.2.2 **Component dependency**. A component developed as part of BUC P invokes a component of BUC Q. This type of dependency is most complex of all, since it calls for early development in an iteration.

5.2.3. **Data dependency**. Records populated by classes in BUC M in a set of database tables are read / updated by classes in BUC N.

Infrastructure components are of two types. One is non-functional and other is functional. Non-functional ones were developed mostly in iteration 1. Business functionality that is used across BUCs needs to be developed early in an iteration. In the course of iteration we also identified all three types of dependencies a priori and coding and testing were prioritized accordingly. This has prevented a number of potential integration issues and made development quite smoother.

6. **Testing Process**

Similar to code integration, testing also happens at various levels for each iteration.

1. **Unit Testing**: Unit testing covers JSPs for their correctness with regard to client side code, and all components involved in the execution of a transaction.

2. **Integration Testing**: The development team tests all components pertaining to a BUC.

3. **System Testing**: Testing team independently tests each BUC from business perspective as a first step. Next, they test all BUCs of that iteration together. In this step, cross BUC functionality gets verified.

4. **Regression Testing**: Components of current iteration are tested along with the components delivered in the previous iterations.

5. **Acceptance Testing**: Finally users do the acceptance testing. This phase has two steps. First step is done independently without migrated data from the legacy system. In the second step application is tested with migrated data.

In the onsite-offshore model first four levels of testing are done at offshore. Unit and Integration testing is done by developers and then application is handed over to the test team. The test team does System and Regression testing independently. This always kept the quality in check before delivery to acceptance testing. Only for interfaces that could not be provided at offshore, system testing is done at onsite. Acceptance testing is at onsite only.
7. Project Management

In this section we elaborate on team structure and organization, iteration planning, traceability from requirements to delivery, effort estimation and risk management.

7.1. Project Organization

We had initially separate teams for architecture, analysis & design, development and testing (See Figure 4). Leads of these teams formed project management team. In the first iteration, construction and testing teams were smaller. Focus was on architecture and design. During later iterations architecture team was made leaner; development and testing teams were augmented. It should be noted that in iterative development, teams and their structure have to be flexible and adaptive than in waterfall.

![Figure 4. Team structure in the first iteration](image)

Initially architecture team addressed deployment and from iteration 2 onwards a separate team was made (See Figure 5). Architecture team dealt with products’ evaluation, installation after procurement, doing ‘proof of concept’, building infrastructure components, and setting up environment for development and deployment. Development team was organized based on BUC and a lead was identified for each BUC. Data migration team also took care of database design & management. This ensured closer monitoring of data migration and application development. Apart from project manager and architect, for all iterations we also had delivery managers who were responsible for estimation and planning for the iteration. The delivery managers reported to the Project manager.

7.2. Iteration Planning

Common question that anyone would encounter in the beginning of the project is on deciding the distribution of use cases across iterations for development. One option is to split the business into slices vertically (See Figure 6). Each slice corresponds to a module of the complete system. For example, in the application we have developed there are modules like Filing of Claims, Adjudication (monetary or non-monetary), Regular Payments, Overpayments, Appeals, Inquiries and Reports. A few BUCs put together can form a module. One way is to build one module after another in iterations. This would give focus on a module during an iteration from development life cycle perspective. The business entities encountered during design and development would be limited to the module. As database design is done subsequently based on classes arrived at, the entities in the database would also be limited to the module.

![Figure 6. Functionality split across iterations](image)
entities in the beginning itself. Attempting to do everything in the first iteration can cause drift in focus from architectural issues to design and development issues. This is one of the possible traps in RUP [17].

Therefore we have taken up one BUC from Filing of Claims and one other from Adjudication for first iteration. Subsequent iterations had a balanced mix up of BUCs from modules to meet the development schedule of other dependent projects and user requirements.

In the case of product development, market conditions play a role on what should be released in which iteration. The ideas mentioned here can be used along with the methodology in [18].

7.3. Traceability

Here we mention traceability from two angles. First one is traceability matrix. All the system test cases are traced to system use cases. All system use cases are traced to business use cases. Second one is ensuring that the knowledge of the system is disseminated uniformly from analysts/designers to developers and testers. Subsequent to analysis, the HTML prototype is used for discussions between designers at onsite, developers and testers at offshore through audio & video conferences. This was also of tremendous help in quickly training the entire team for complete functionality. Otherwise, a BUC team is less likely to know the dependencies and testers would not be able to get the full picture of the requirements. Subsequently the material of these sessions was reused to train all users before going live.

7.4. Effort Estimation

Reifer [19] proposes an estimation model for web development projects in general. We present much simpler model that can be used after the customer signs off the HTML prototype. As mentioned earlier basic ‘unit of work’ is a transaction. From iteration 2 onwards we classified transactions into four types. On click of a button,

1. Response is thrown from servlet itself, without any processing. ‘Help’ is an example.
2. Some processing is done by the servlet with a helper or utility class. Typically ‘cancel’ or ‘close’ type of buttons, which need session cleanup.
3. Read operations are performed on database.
4. Write or update operations are performed on database.

Based on the effort spent in iteration 1 and 2, we found the average effort to be spent for a type of transaction. From iteration 3 onwards we used the guideline to estimate the effort for subsequent development as soon as the HTML prototype is completed at onsite. This helped us in more accurate planning of the schedule and resources. This could not have been done if we were to follow waterfall.

7.5. Risk Management

One of the top reasons for adopting RUP is early risk mitigation, in comparison with Waterfall model. However, the risks and their mitigation are left to the project team. The risk analysis presented by Deck [6] is based on over 100 criteria. We followed a simple and effective idea. We had implemented a fine grained risk assessment and mitigation technique by absorbing it as part of the development process. During design activity HTML prototype is made and discussed with the users. The possible new transactions in the application and their types are completely known after the prototype is frozen. Their risk is assessed taking into account business priority, transaction type, development and integration. Most risky transactions are targeted first, followed by less risky ones and so on.

8. Data Migration

The legacy data was in VSAM and IDMS tables. At the time of database design, data migration aspects were also taken into account. The application owners at onsite provided mapping rules from legacy database to new database. Testing of data migration and application were done independently. Subsequently, the application was tested along with migrated data.

9. Conclusion

We summarize the generalizations here:

- Iterative development enables process improvement and maturity in the first few iterations of the same project.
- Normally there exist three views for a web application: User Interface View (Transactional), Business Objects View (Object Oriented) and Data View (mostly Relational). Transformation of data from one view to another needs to be addressed in design.
- Continuous integration helps in identifying integration issues immediately. Incremental
introduction delays identification of bottlenecks to the end of an iteration.

- Code dependencies are three types: Screen dependency, Component dependency and Data dependency. These should be addressed first in iteration.

- Unit testing all J2EE components (JSP, Servlet and EJB) with stubs can delay integration issues between components executed in a transaction. The benefit from the usage of stubs needs to be assessed a priori.

- While executing a large project in iterative model, team structure needs to be flexible and adaptive.

- Risk management should be integrated with development process.

In the paper we have elaborated on the implementation of RUP in the context of Onsite-Offshore model with ISO and CMM quality processes. We have also brought out the three faces of J2EE applications and how they need to be managed. We have analyzed the levels of integration and testing and highlighted that continuous integration reduces the elapsed time for development. An effort estimation model based on type of transactions is presented. Our experience in Project organization, Iteration planning and traceability is explained. Finally we successfully implemented a fine grained risk management concept that identifies risk to the lowest level of 'unit of work'.

References


[14] A. Subrahmaniam, Separation of Behavior and Data – OO or Not? At URL http://www.subrahmanyam.com/notes/NotePad.html#BehaviorAndData


