An Agile Methodology for Reengineering Object-Oriented Software

Anam Sahoo, David Kung, and Sanika Gupta
Department of Computer Science and Engineering, The University of Texas at Arlington, USA

***Abstract*— Software maintenance is an important phase in the software development life cycle. More than 75% of maintenance efforts are enhancement. Currently, most enhancement projects are carried out in an ad hoc manner, depending on the knowledge and experience of the developers. Software reengineering aims to provide an engineering approach for software enhancements. In this paper, we present an agile reengineering methodology for object-oriented software. The methodology has a quick planning phase followed by a series of iterative reengineering phases. Each iteration consists of three legs: the reverse engineering leg, the reincarnation leg, and the validation leg. Academic and industry experiments show promising results.**

Keywords and phrases: Software process and methodology, software maintenance, software reengineering, agile method, object-oriented software.

I. INTRODUCTION

Software maintenance typically consumes an average of 60% of software life costs, with enhancements being responsible for more than 75% of the costs [11]. These costs are a grand challenge for the current software community, in which tens of millions of lines of legacy code need to be modified during enhancement maintenance. The problem becomes even direr when the enhancement project is performed by engineers who do not have sufficient knowledge of the legacy system and documentation is inadequate or nonexistent. Software reengineering aims to provide an engineering approach for software enhancement. Current literature surveys reveal that there is a lack of a systematic reengineering methodology.

In this paper, we present a methodology for reengineering object-oriented software. It has three distinct phases: a release planning phase, iterative reengineering phase, and a system validation phase as shown in Fig. 1.

Each release begins with a quick agile planning phase, followed by an iterative reengineering phase consisting of a series of iterations. At the end of the release, an optional system validation phase is performed to validate the release before it is formally delivered to customers. The planning phase performs two activities. First, new requirements are identified and prioritized by applying information collection techniques and are based on a statement of work (SOW) from the customer. Second, new use cases and changes to existing use cases are derived. Finally, planning for release iterations is performed to produce a roadmap to guide the iterative reengineering activities. The iterative reengineering phase consists of a series of iterations. Each iteration has three legs: the reverse engineering leg, the reincarnation leg, and the validation leg as shown in Fig 2. This is referred to as the N-process model.

The reverse engineering leg recovers design artifacts and helps to understand the existing system. It starts from a legacy code and has three major outputs: recovered design, recovered architecture, and recovered requirements in the form of use cases. There are techniques described in

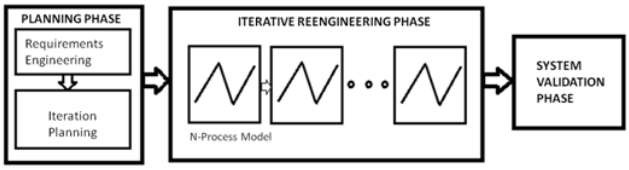

Figure 1. Agile OO SW reengineering methodology

[2,6,12,17-18,21,23,25,28,33,35] for recovering these design artifacts. The middle leg is the re-incarnation leg, which transforms the legacy system to a new working system. The third leg of the iterative reengineering phase focuses on validating the implementation against the intended design and requirements by preparing appropriate test cases. These include component level unit test cases, subsystem/system integration test cases, and system test cases. The system validation phase is meant to perform a formal release testing even though functional unit and integration testing have already been performed during the iterative reengineering phase. A formal system testing is conducted when a release candidate build is ready. Then a well tested release build is handed over to customer for customer acceptance testing.

The detailed step-by-step methodology for these phases is described in section II. Section III describes the application of the methodology to two academic experiments, which show significant improvements in project schedule and software quality. Section IV describes related work, followed by conclusions and future work in section V.

II. THE REENGINEERING METHODOLOGY

To illustrate the steps of the methodology, the Academic Advising Scheduler Web (AASW) system will be used as the legacy system. It is a software application written in Java and JSP with a MYSQL data base. It supports three types of users: administrators, advisors, and students. TABLE I lists the existing legacy requirements and use cases assumed to be already available to limit the scope of this paper. If missing, the needed use cases, can be recovered using techniques described in [6,43].

A. Planning Phase:

The planning phase performs two activities. First, new requirements are identified and prioritized by applying information collection techniques. Their impact on existing use cases is assessed, resulting in new, modified, and deleted

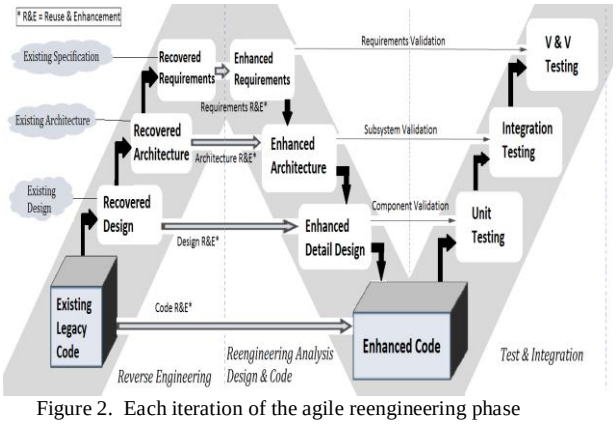

Figure 2. Each iteration of the agile reengineering phase

DOI reference number: 10.18293/SEKE2016-227

use cases, respectively. Second, planning for release iterations is performed to produce a roadmap to guide the iterative reengineering activities.

Identifying and Prioritizing Enhancement Requirements: Requirements are part of the contract of a software project. They specify the capabilities that the software system must deliver. Thus, correctly identifying and prioritizing enhancement requirements are critical to the success of a reengineering project. To identify and prioritize enhancement requirements, information collection techniques such as customer presentation, user survey, user interview, and literature survey are used. Next, new, modified and deleted use cases are derived. A new use case is derived if an application-specific verb-noun phrase is found or inferred from an enhancement requirement and the verb-noun phrase satisfies the following conditions: (1) it denotes a complete end-to-end business process of the application, (2) the business process begins with a user, (3) the business process ends with the user, and (4) the business process accomplishes a useful business task for the user. Sometimes, an enhancement requirement specifies that a piece of functionality needs be incorporated to an existing use case of the legacy system. Finally, existing use cases may no longer be needed due to the changing business environment. *For* AASW, the enhancement requirements identified during the planning phase are: R4: Users can change passwords. R5: Users must change system-generated temporary passwords when logs in for the first time. R6: Student ID and passwords must be encrypted before storing them in the database. From R4 above, we derive one verb-noun phrases: change password. This satisfies the four conditions for a use case described above. Therefore, a "Change password" new use case is derived. From both R5 and R6 above, we derive four modified use cases: "UC1: Create advisor," "UC6: Create student," "UC4: Login user," and "UC7: Schedule appointment." The existing use cases, UC1 and UC6 need modifications to incorporate encryption of temporary password. The UC4 needs modification to force the user to change the temporary password during login and encrypt the new password. The UC7 should be modified to include encryption of student ID while scheduling an appointment. Encryption of the password will be implemented in the new UC8: change password use case. TABLE II shows the impact to use cases. R4-R6 have priority 1, and must be completed in the first reengineering iteration.

The new, modified, and deleted use cases are assigned to iterations in this step. First, an agile estimation technique such as the poker game [7] is applied to obtain an effort estimate for dealing with each of the use cases. An order to design, implement, delete, and test the use cases is derived, based on their dependencies and priorities. The use cases are then assigned to iterations according to the order.

TABLE I. REQUIREMENTS AND USE CASES FOR THE EXISTING LEGACY SYSTEM

Legacy Requirements	Legacy Use Cases
R1: An Administrator can create, edit and delete advisors and define their privileges	UC1: Create advisor UC2: Edit advisor UC3: Delete advisor
R2: Advisors can login and specify their advising time.	UC4: Login user UC5: Update schedule
R3: Student can create account, login to it, and schedule an appointment with an advisor.	UC6: Create student UC4: Login user UC7: Schedule appointment

B. *Iterative Reengineering Phase:*

The iterative reengineering phase consists of a series of iterations. Each iteration has three legs: the reverse engineering leg, the reincarnation leg, and the validation leg. It is worth noting a new term "reincarnation" here for the middle leg of the N-process model instead of using the over used term "reengineering" to avoid confusion. We interpret reengineering as a complete end-to-end methodology. Reverse engineering is performed only if design documentation that accurately represents the code does not exist. The reverse engineering leg starts from a legacy code and has three major outputs: recovered design, recovered architecture, and recovered requirements in the form of use cases. First, design artifacts such as class diagrams, sequence diagrams, and use cases can be recovered from existing code. High-level architectural design, domain model, and requirements for legacy system can then be derived from the recovered design artifacts. TABLE III summarizes all the needed artifacts and activities for each of new, modified, and deleted use cases.

In this paper we discuss only two reverse-engineered artifacts--- that is, implementation sequence diagram (ISD) and implementation class diagram (ICD). An ISD shows how the software objects interact with one another and in what order to process a user request. An ICD is an integrated view of the implemented classes, their attributes, methods, and relationships. Detailed steps for reverse engineering ISD and ICD are given below.

B.1. Reverse engineering ISD and ICD:

Step 1. Observe how a user uses the current system and describe the actor system interaction behavior. For example, a login use case behavior is as follows: (a) AASW displays any page having username and password fields in the page header area, (b) the user enters username and password and clicks "Submit," (c) if the user authenticates correctly, AAWS displays the user's dashboard, and (d) the user sees his dashboard.

Step 2. Identify nontrivial step(s) as follows: (a) If the step does not require background processing, or (b) if the system response simply displays a menu/input dialog, or (c) if the step displays the same system response for all actors, then it is a trivial step. but (d) if the system response is different for different actors, then it is a non-trivial step. In above example, the nontrivial step is (d).

TABLE II. ENHANCEMENT REQUIREMENTS AND IMPACT TO USE CASES

Enhancement Requirements	Use Cases	Category
R4: Users can change passwords.	UC8: Change password	New
R5: Users must change system-generated temporary passwords when logs in for the first time.	UC4:Login user UC8:Change password	Modified New
R6: Student ID and all passwords must be encrypted before storing them in the database.	UC1:Create advisor UC6:Create student UC4:Login user UC7:Schedule appointment UC8:Change password	Modified Modified Modified Modified New

TABLE III. REENGINEERING ARTIFACTS AND ACTIVITIES FOR DIFFERENT USE CASE CATEGORY

Artifacts/Activities	Category of Use Case		
	New	***Modified***	***Deleted***
Reverse engineer ISD	Not needed.	Yes, needed, to be modified to take into account the enhancement requirements.	Yes, needed, to identify potential classes and methods to delete.
Reverse engineer ICD	Yes, needed, to identify classes and methods to reuse or extend	Yes, needed, to identify classes and methods to reuse or extend.	Yes, needed, to identify classes and methods to delete.
Construct expanded Use Case (EUC)	Yes, needed.	May be needed if actor-system interaction behavior need be changed.	No, not needed
Construct/modify Design Sequence Diagram (DSD)/ISD	New DSD - with new and existing classes from ICD. May apply software design patterns (SDP) such as adapter, controller, and facade.	Modified ISD, consider reusing existing classes from ICD.	Use ISD to identify classes and methods to delete.
Modify ICD	Add new classes and modify classes of ICD according to the DSD. Enhance design with SDP.	Add new classes and modify classes of ICD according to modified ISD. Enhance design with SDP such as adapter and facade.	Delete identified classes and methods from ICD.
Create New Test Cases	Needed for new classes and new methods, and classes affected by changes.	Needed for modified classes and methods, and classes affected by the changes.	No new test cases needed
Do Regression Test	Yes, needed.	Yes, needed.	Yes, needed.

Step 3. Identify the button that initiates the nontrivial step. Identify the action listener for the button. Trace the action listener handler code for objects and messages sent between them. Construct implementation sequence diagrams (ISD) to describe interactions between these objects. Techniques and tools for reverse engineering code to produce ISD and ICD are found in [13,17-18,21,23,25-26] and [13,18,22,26-28,30] respectively. Some of these tools could be used to reduce effort.

Reincarnation: This middle leg transforms the legacy system to a new working system. The new, modified and deleted use cases are already identified during planning phase. For each use case, depending on its category, the following three steps are performed: First, an individual use case is used to identify and recover implemented design and high-level architectural artifacts from the code. Second, the necessary additions, modifications, and deletions are made to those recovered artifacts. Finally, changes in the artifacts are incorporated into the existing code during the implementation phase.

Validation: The third leg of the iterative reengineering phase focuses on validating the implementation against the intended design and requirements by preparing appropriate test cases. These include component level unit test cases, subsystem/system integration test cases, and system test cases. The combined detailed steps for reincarnation and validation activities for new, modified, and delete use cases are as follows:

B.2.Reengineering for a New Use Case:

Treatment for new use cases is similar to forward engineering, except that existing the legacy code and some of the test cases may be reused. Forward engineering is described in various publications [1,3,14-16,18]. The steps are summarized as follows:

Step 1. For each new use case, describe how a user or actor will interact with the system to carry out the business process. This is called actor-system interaction modeling/design. Consider, for example, the "change password" use case as described in the planning phase A.1. It may go as follows: (a) the user select the "change password" tab after login, (b) the system asks the user to enter information: new pass word, and confirmation password, (c) user enter needed information and press enter, (d) the system displays a confirmation message, and (e) the user sees the confirmation message that the new password is successfully stored in the system.

Step 2. Identify actor-system interaction steps that required background processing such as database access or user-dependent computation. For example, steps (c) and above require the system process user information and display the confirmation message to the user. We call such steps nontrivial steps.

Step 3. For each pair of nontrivial steps, produce a design sequence diagram (DSD) to describe how software objects would interact with each other through message passing (or function calls) to produce the output (e.g., the confirmation message) from the user input (e.g., new password and confirmation password). Software reengineering should reuse existing code as much as possible, classes of the ICD as recovered in section B.1. Extract classes from the DSDs along with their methods, attributes, and relationships such as call relationship and use relationship. Use these to modify the recovered ICD as follows. To facilitate identification of classes, methods, and relationships to be implemented, the changes are highlighted in the ICD accordingly. (a) If a class is not in the ICD, then add the class to the ICD along with all its methods and attributes extracted from the DSDs. (b) If a class is in the ICD but some of its methods/attributes extracted from the DSD are not in the ICD, then add these to the class in the ICD. (c) If a relationship is not in the ICD, then add the relationship to the ICD.

Step 4. Implement new classes and methods and modify the existing classes and methods as per the new DSD and modified ICD.

Step 5. Prepare and execute functional unit and integration regression test cases for new classes and methods, modified classes, and classes and methods affected by changes.

Step 6. Make the necessary code changes until the test cases successfully pass as per TDD.

Step 7. Maintain traceability to keep track of changes. Examples of DSD and ICD for "UC8: Change password" new use case for AASW code base A are shown in Fig. 3 and Fig. 4 respectively.

B.3.Reengineering for a Modified Use Case:

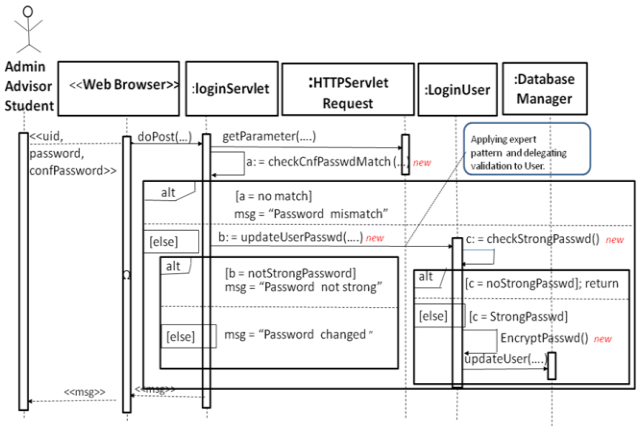

Figure 3. Change password DSD

Reengineering for modified use cases is the most common reengineering activity. The steps are as follows:

Step 1. Observe how a user uses the current system and reverse engineer the ISD and ICD as specified in the above reverse engineering section B.1.

Step 2. Modify the recovered ISD as required by the enhanced requirements. This step should attempt to reuse legacy code classes and is described in step 3 for new use cases in section B.2.

Step 3. Extract classes, methods and relationships from the modified recovered ISD and use them to modify the ICD, as described in step 4 for new use cases in section B.2.

Step 4. For each class method, prepare necessary unit test cases and subsystem/system integration regression test cases in parallel while incorporating the necessary code changes. Perform the code review, functional unit testing and integration regression testing immediately before and after any code modification is performed as per TDD.

Step 5. Maintain traceability to keep track of changes. Recovered/modified ISD for UC4 for code base A is shown in Fig. 5.

B.4.Reengineering for a Deleted Use Case:

Deleted use cases need to be handled carefully as follows: *Step 1.* Observe how a user uses the current system and reverse engineer the ISD as specified in the above reverse engineering section B.1.

Step 2. Identify the classes and methods of the ICD that need to be deleted.

Step 3. Identify and run necessary regression integration test cases for each deleted class and run before deleting any code. Check that functionality exists.

Step 4. Comment out classes and methods and make sure that no other modules are dependent on them. Comment out classes or methods that are not used.

Step 5. Run the system/subsystem integration regression test cases immediately after the code is commented out as per TDD. Make sure that the functionality is deleted. Identify and run selective impacted other regression test cases.

Step 6. Delete commented classes and methods and make necessary changes to the ICD.

Step 7. Maintain traceability to keep track of changes.

III. CASE STUDIES

In this section, we briefly present the impact of the methodology on schedule and quality of reengineering projects in comparison to doing it in any other way.

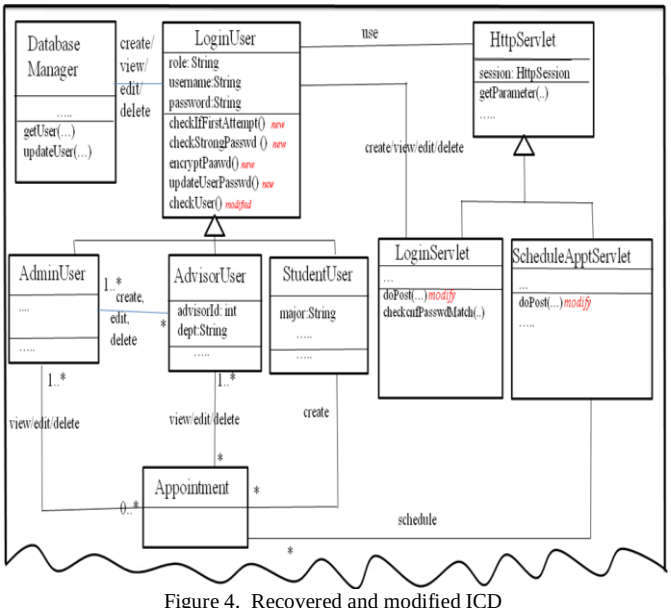

Figure 4. Recovered and modified ICD

Overview of Case studies: We conducted two academic case studies, as summarized in TABLE IV and an industry one. The academic case studies involved 30-31 graduate students in two different semesters using two legacy code bases of the AASW system. The third case study is for an industry project conducted by a local railroad company to reengineer a Driver-Assist legacy system to an Auto-pilot system involving a team of 8 experience engineers. For the first phase of the first case study, students divided into 7 teams were asked to do their reengineering assignments in their own ways. Then, the methodology was taught to the class. In the second phase, the code bases were swapped and asked to do the second reengineering assignments following the methodology. In the second case study, the eight teams were asked to learn and use the Rational Unified Process (RUP)[4] and Agile Unified Methodology (AUM)[16] for the first assignment. Then, teams used the methodology for the second assignment after swapping the code bases. For both the phases of case studies, data were collected from all the teams in the following two ways: (1) artifacts submission that include enhanced code and (2) the teams were asked to give demonstration of their working code to

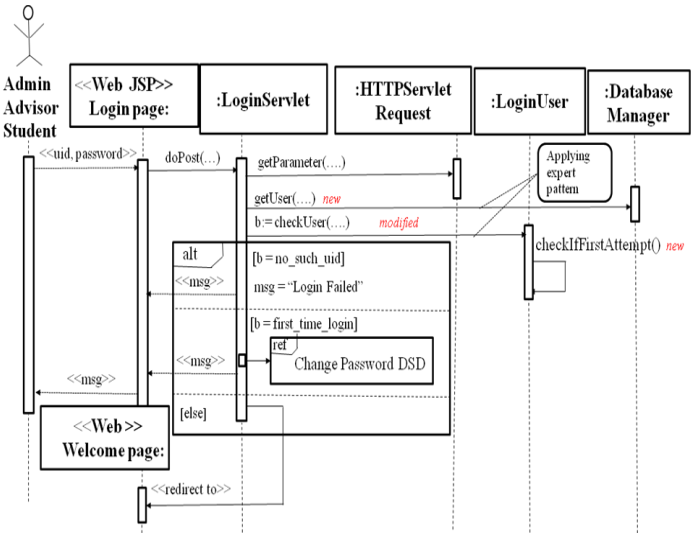

Figure 5. Login user recovered and modified ISD

TABLE IV. OVERVIEW OF CASE STUDIES

Case Studies		1		2	
# of Participants		30 CSE Students		31 CSE Students	
Teams		2-4	1, 5-7	1,3,5,7	2,4,6,8
AAWS Project code base	Before learning the methodology	A	B	A	B
	After learning the methodology	B	A	B	A
	Duration	5 weeks	5 weeks	5 weeks	5 weeks

validate their claim. For the third case study, the railroad company used and shared their experience of using the methodology.

Analysis of the data collected from the first case study was performed by comparing different groups for the same legacy systems--that is, the quality of the enhanced code base A performed by the teams 2-4 in case study 1 using the methodology was compared with 3 out of the rest of the 4 teams (randomly picked) using ad hoc ways. But in the second academic case study, the analysis was performed comparing student performance with RUP and AUM with that of the methodology.

Analysis and Results Interpretation: Case study 1 results indicate that for both the code bases of AASW, following any ad hoc way, only one out of three teams submitted and demonstrated the working code. All the teams for both code bases just submitted the requirements as it was given to them without any prioritization, categorization, effort estimation, and traceability. In contrast, the teams who followed the methodology for both the code bases performed much better with 78-80% of enhancement requirements completed. All the teams submitted all the listed artifacts, which were evaluated to be of much better quality. The second case study results intdicate that students following RUP or AUM could complete only 30 to 50% of needed enhancements. The team who submitted the working code just submitted some ISD and ICD but no other artifacts. In contrast, the teams performed much better with 75-80% enhancement completion as shown in TABLE V. All the teams submitted all the listed artifacts, which again were evaluated to be of much better quality. Feedback from the industrial application indicated that the methodology significantly imroved the schedule and quality of the company's safety critical auto-pilot reengineering project in terms of requirements understanding, reverse engineering, enhancement of design, implementation, and testing. The estimated improvements was perceived to lead to a 25% reduction in schedule due to smooth integration and a 50% reduction in defects as compared to similar past reengineering projects.

There are a few limitations to our first two case studies: first, only two AASW legacy code bases were used to compare the effectiveness of the methodology. Second, even though students are inexperienced and the two projects have different code bases, the domain learning during the first assignment was definitely an advantage for the second assignment. Finally, the sample size is too limited to run any statistical analysis on. Despite all these limitations, these case studies show preliminarily that the participants performed much better using the proposed methodology compared to using RUP, AUM or doing it in any ad hoc way.

IV. RELATED WORK

There are very few studies performed on reengineering processes and methodology even though several plan-based and agile forward engineering processes and methodologies are currently used for reengineering activity. A framework-based agile reengineering process named PARFAIT using static structure of rational unified process (RUP) is described in [4], which explains how to rapidly provide an user with evolved versions of legacy system. [5] describes a segmentation reengineering process after recovering the analysis model from the procedural legacy C code and then partially transforming it to an object-oriented java code using design patterns. Another reengineering process for migrating legacy object-oriented systems to component based systems is described in [19], which suggests process metrics to improve code granularity and reusability. [32] explains an ontology based approach to reengineer legacy enterprise software to cloud computing environment. A reengineering process called "The Renaissance" is overviewed in [3]. It is a two-stage process for transforming legacy system to

TABLE V. PROJECT ARTIFACTS AND CODE COMPLETION STATUS COMPARISON

Code Base	**Submitted Artifacts**	**Case Study 1**								**Case Study 2**									
		Teams using any Ad-hoc way				**Teams using The Methodology**				**Teams using process**					**Teams using the Methodology**				
										RUP		**AUM**							
	Teams →	***2***	***3***	***4***	***Avg.***	***1***	***5***	***6***	***Avg.***	***1***	***3***	***2***	***4***	***Avg.***	***5***	***6***	***7***	***8***	***Avg.***
Base - A	*Requirements*	x	x	x	100%	x	x	x	100%	x	x	x	x	100%	x	x	x	x	100%
	ISD/DSD	-	-	-	0%	x	x	x	100%	-	-		x	25%	x	x	x	x	100%
	ICD	-	-	x	33%	x	x	x	100%	-	-	x	-	25%	x	x	x	x	100%
	Test cases	-	-	-	0%	x	x	x	100%	-	-	-	-	0%	x	x	x	x	100%
	Traceability	-	-	-	0%	x	x	x	100%	-	-	-	-	0%	x	x	x	x	100%
	Running code	-	-	x	33%	x	x	x	100%	-	x	x	x	75%	x	x	x	x	100%
	Completion %	0	0	10	3 %	60	95	90	82%	0	40	30	35	24%	70	60	80	90	80%
	Teams →	***5***	***6***	***7***	***Avg.***	***2***	***3***	***4***	***Avg.***	***5***	***7***	***6***	***8***	***Avg.***	***1***	***2***	***3***	***4***	***Avg.***
Base - B	*Requirements*	x	x	x	100%	x	x	x	100%	x	x	x	x	100%	x	x	x	x	100%
	ISD/DSD	-	-	-	0%	x	x	x	100%	-	-	x	x	50%	x	x	x	x	100%
	ICD	x	-	-	33%	x	x	x	100%	-	-	x	-	25%	x	x	x	x	100%
	Test cases	-	-	-	0%	x	x	x	100%	-	-	-	-	25%	x	x	x	x	100%
	Traceability	-	-	-	0%	x	x	x	100%	-	-	-	-	25%	x	x	x	x	100%
	Running code	x	-	-	0%	x	x	x	100%	-	x	x	x	25%	x	x	x	x	100%
	Completion %	33	0	0	11%	65	95	75	78%	0	0	25	50	19%	60	70	90	80	75%
" x " – Artifacts submitted and "-" – Artifacts not submitted .																			

evolvable system: first, the strategic planning stage, and then, the continuous evolution stage.

Missing or outdated documentation in legacy projects is always an issue during reengineering. Many techniques, however, have been presented for reverse engineering artifacts, such as domain models[12], class diagrams[28], sequence diagrams [16,18,23,25,33], and use cases[7] from legacy code. An architecture recovery methodology using feature modeling is described in [24]. In this methodology, the top-down architectural element hypotheses are generated based on domain knowledge and verified using bottom-up tracing procedures. Finally, feature models are introduced bridging the gap between requirements and architecture. The rapidly changing business environment causes requirements to constantly change. However, missing legacy requirements or use cases is a common problem, and recovery is a very complex affair. A few use case recovery techniques are described in [6, 34]. The most important of all of these is the ability to trace all the reengineering activities from legacy code to requirements, to reincarnated design elements, and enhanced code. Tracing all around in reengineering using RETH tool is described in [10].

V. Conclusions and Future Work

Reengineering is an important part of software maintenance in an industry in which the environment is constantly evolving and customer needs are ever-changing. This paper presents an agile methodology to reengineer object-oriented software, which focuses on a front end quick planning phase, an iterative development phase, and a system validation phase using test driven development approach. The application of the methodology on academic and industry experiments gives an early indication of improved code quality and project schedule over using RUP, AUM, or doing it in any ad-hoc way. The future work that remains to be completed seeks to extract a domain model and use cases from the recovered ICD and ISD iteratively. The agility of the methodology can also be improved by automating the manual steps and integrating them with existing reverse engineering tool sets. Finally, the code can be enhanced by restructuring with design patterns.

References

[1] M.R. Blaha & J.R. Rumbaugh, "Object-Oriented Modeling and Design with UML (2nd Edition)," Prentice Hall, 2004.

[2] J. Borchers, "Invited Talk: Reengineering from a Practitioner's View -- A Personal Lesson's Learned Assessment," 15th European Conference on Software Maintenance and Reengineering (CSMR), 2011., pp. 1-2.

[3] B. Bruegge & A.H. Dutoit, "Object-Oriented Software Engineering: Using UML, Patterns, and Java (3rd Edition)," Prentice Hall, 2009.

[4] M.I. Cagnin & J.C. Maldonado, "PARFAIT: towards a framework-based agile reengineering process," Proceedings of the Agile Development Conference (ADC), 2003., pp. 22-31.

[5] M.I. Cagnin, R. Penteado, R.T.V. Braga & P.C. Masiero, "Reengineering using design patterns," . Proceedings Seventh Working Conference on Reverse Engineering, 2000., pp. 118-127.

[6] F. Chen, H. Zhou, H. Yang, M. Ward & W.C.C. Chu, "Requirements Recovery by Matching Domain Ontology and Program Ontology," IEEE 35th Annual Computer Software and Applications Conference (COMPSAC), 2011., pp. 602-607.

[7] P. Claudia, M. Liliana & F. Liliana, "Recovering Use Case Diagrams from Object Oriented Code: An MDA-based Approach," Eighth International Conference on Information Technology: New Generations (ITNG), 2011., pp. 737-742.

[8] M. Cohn, "Agile estimating and planning Prentice Hall Professional Technical Reference, 2006.

[9] J.M. deBaud & S. Rugaber, "A software re-engineering method using domain models," International Conference on Software Maintenance, 1995., pp. 204-213.

[10] G. Ebner & H. Kaindl, "Tracing all around in reengineering," IEEE Software, vol. 19, no. 3, 2002., pp. 70-77.

[11] R.L. Glass, "Frequently forgotten fundamental facts about software engineering," IEEE Software, vol. 18, no. 3, 2001., pp. 112-111.

[12] M. Hong, T. Xie & F. Yang, "JBOORET: an automated tool to recover OO design and source models," 25th Annual International Computer Software and Applications Conference (COMPSAC), 2001., pp. 71-76.

[13] IBM tool, "Rational Rose Architectl," Available: http://www.ibm.com/software/products/en/ratisoftarch.

[14] I. Jacobson, J. Rumbaugh & G. Booch, "Unified Software Development Process Addison-Wesley, 1999.

[15] D.C. Kung, "On use case identification," , Boston, USA, Proc. of 25th Int'l Conf. on Software Engineering and Knowledge Engineering, June 26-29, 2013.

[16] D.C. Kung, "Object-oriented software engineering: an agile unified methodology (International Student Edition)," McGraw-Hill, a business unit of the McGraw-Hill Companies, Inc, 2014.

[17] Y. Labiche, B. Kolbah & H. Mehrfard, "Combining Static and Dynamic Analyses to Reverse-Engineer Scenario Diagrams," 29th IEEE International Conference on Software Maintenance (ICSM), 2013., pp. 130-139.

[18] C. Larman, "Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition)," Prentice Hall, 2005.

[19] E. Lee, B. Lee, W. Shin & C. Wu, "A reengineering process for migrating from an object-oriented legacy system to a component-based system," The 27th Annual International Conference on Computer Software and Applications(COMPSAC), 2003., pp. 336-341.

[20] M. Lee & S. Park, "A methodology to extract objects from procedural software," The 24th Annual International Conference on Computer Software and Applications(COMPSAC) 2000., pp. 557-566.

[21] L. Martinez, C. Pereira & L. Favre, "Recovering sequence diagrams from object-oriented code: An ADM approach," International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE), 2014., pp. 1-8.

[22] Object-Aid tool, "ObjectAid," Available: http://objectaid.com/.

[23] T. Parsons, A. Mos, M. Trofin, T. Gschwind & J. Murphy, "Extracting Interactions in Component-Based Systems," IEEE Transactions on Software Engineering, vol. 34, no. 6, 2008., pp. 783-799.

[24] I. Pashov & M. Riebisch, "Using feature modeling for program comprehension and software architecture recovery," The 11th IEEE International Conference and Workshop on Engineering of Computer-Based Systems, 2004., pp. 406-417.

[25] A. Serebrenik, S. Roubtsov, E. Roubtsova & M. van den Brand, "Reverse Engineering Sequence Diagrams for Enterprise JavaBeans with Business Method Interceptors," The 16th Working Conference on Reverse Engineering, WCRE, 2009., pp. 269-273.

[26] Sparxsystems tool, "Enterprise Architect," Available: http://www.sparxsystems.com/.

[27] Tigris tool, "AgroUML," Available: http://argouml.tigris.org .

[28] P. Tonella & A. Potrich, "Static and dynamic C++ code analysis for the recovery of the object diagram," International Conference on Software Maintenance, 2002., pp. 54-63.

[29] M. Trudel, C.A. Furia, M. Nordio, B. Meyer & M. Oriol, "C to O-O Translation: Beyond the Easy Stuff," The 19th Working Conference on Reverse Engineering (WCRE), 2012., pp. 19-28.

[30] UML-Lab tool, "UML-Lab," Available: http://www.uml-lab.com/en/uml-lab/academic/.

[31] I. Warren & J. Ransom, "Renaissance: a method to support software system evolution," The 26th Annual International Computer Software and Applications Conference (COMPSAC), 2002., pp. 415-420.

[32] H. Zhou, H. Yang & A. Hugill, "An Ontology-Based Approach to Reengineering Enterprise Software for Cloud Computing," IEEE 34th Annual Computer Software and Applications Conference (COMPSAC), 2010., pp. 383-388.

[33] T. Ziadi, M.A.A. Da Silva, L.M. Hillah & M. Ziane, "A Fully Dynamic Approach to the Reverse Engineering of UML Sequence Diagrams," 16th IEEE International Conference on,Engineering of Complex Computer Systems (ICECCS), 2011., pp. 107-116

[34] Q. Li, S. Hu, P. Chen, L. Wu & W. Chen, "Discovering and Mining Use Case Model in Reverse Engineering," Fourth International Conference on Fuzzy Systems and Knowledge Discovery(FSKD), 2007., pp.431-436