International Journal of Information & Computation Technology.
ISSN 0974-2239 Volume 4, Number 7 (2014), pp. 681-686

http://www. irphouse.com

An Object Oriented Design Approach for Modification of Rotten Code using Regression Testing & Refactoring

Anita Arora[1] and Naresh Chauhan[2]

[1,2]Computer Engineering, YMCA University of Science & Technology, Faridabad, Haryana, INDIA

ABSTRACT

Design is the crucial step while working on any user story during the sprint in an agile culture of software development. A good design can generate good code and moving further in the journey, a good code would have less or minimum bugs by utilizing benefits of various principles of agile like simplicity, pair programming, less is more approach etc. to its fullest. In this paper, a step wise source code design approach has been proposed for the purpose of obtaining improved code from rotten code using regression testing and refactoring/rewriting methodology. Definition of improved code is proposed on the basis of various design principles like open close principle, dependency inversion principle, interface segregation principle, single responsibility principle and liskov's substitution principle. This improved design of code executes the same behavior irrespective of the change in the design feature of the original code. The method used for the proposed conversion comprises: defining one or more preconditions for a source code refactoring conversion, applying the refactoring transformation to source code and providing a user with results after the at least one precondition is tested. The presence of critical errors in the previous steps slows the performance of refactoring process. That's why, a regression test need to be performed at every step of the sprint and precondition need to be tested for better performance.

Keywords- Agile; Object Oriented Design; Refactoring; Regression Testing; Rotten code

1. INTRODUCTION

Agile software development is a goal oriented game in which teammates are the players and by 3C (collaboration, continuous integration and cooperation) approach of

agile, software/small increments are released to the customer. In this approach, new changes are accommodated during the sprint/iteration. Software professionals who have been in the software industry for a few years know the feeling of work pressure when they are faced with fixing a defect or implementing a new feature in the existing user story which is poorly designed. This type of code cannot be handled by software workers during maintenance face as it smells because of its rotten behavior. A good design in software development is important criteria for the success of a project.

Extreme programming (XP), one of several emerging, so called agile methodologies attempts to convert the software development life cycle process into a modified process to improve code quality and fundamental design of the software under development with its unique features. XP is practiced through simple design, small releases, refactoring, pair programming, test driven/design development (TDD) and continuous integration. Out of all these pair programming, TDD and refactoring are need of the time. Refactoring is achieved by removing duplicates, simplifying structure, retaining the original behavior and adding extensibility/scalability. Further, pair programming way of working ensures that code review is performed side by side so as to have good quality code with minimum or less bugs. This style of working is tough to follow in the beginning but it's afterwards results are awesome.

TDD is comprised of design and programming activity, not testing activity per se. It includes designing of failing test, implementing code to pass the test and improve the design by refactoring. Designing activity which is part of TDD here is the starting step for getting the right outcome of the software. So, there is a need to focus on creating good design instead of cumulative bad design from first stage to last stage. This will incur more defects, more cost, more time and more resources. Simple design expertise in XP can be achieved by following practice of implementing design principles of object oriented paradigm such as open close principle (OCP), dependency inversion principle (DIP), interface segregation principle (ISP), single responsibility principle(SRP) and liskov's substitution principle(LSP). Here, good design/software is measured with efficacy. Efficacy of any software can be judged on the basis of two factors namely technical advancement of existing design and secondly it's economic value in terms of market standards.

In this paper, a refactoring approach for improving design has been proposed by considering regression testing which has higher quality as compared to existing refactoring methods in an agile environment. Also, syntactic and semantic checks would be performed so as to ensure consistent behavior of the user story. With this approach scalability/extensibility chances would be higher. This paper has been further divided into different sections. Section 2 of this paper recites literature survey, Section 3 is about proposed model for the improved code, Section 4 discusses the automated tool for refactoring c/c++/Java source code and last section concludes the paper.

2. LITERATURE SURVEY

2.1 Bad Design

Software design principles represent a set of guidelines that helps us to avoid

having a bad design. The design principles are associated to Robert Martin who gathered them in "Agile Software Development: Principles, Patterns, and Practices". According to Robert Martin there are 3 important characteristics of a bad design that should be avoided:

- *Rigidity* It is hard to change because every change affects too many other parts of the system.
- *Fragility* When you make a change, unexpected parts of the system break.
- *Immobility* It is hard to reuse in another application because it cannot be disentangled from the current application.

2.2 Object Oriented Design Principles

- *Open Close Principle (OCP)* Software entities like classes, modules and functions should be open for extension but closed for modifications.
- *Dependency Inversion Principle (DIP)* High level modules should not depend on low level modules. Both should depend on abstractions. Abstractions should not depend on details. Details should depend on abstractions.
- *Interface Segregation Principle (ISP)* Clients should not be forced to depend upon interfaces that they don't use.
- *Single Responsibility Principle (SRP)* A class should have only one reason to change.
- *Liskov's Substitution Principle (LSP)* Derived types must be completely substitutable for their base types.

3. PROPOSED MODEL

Regression testing ensures that value is delivered in the code after applying manual or automated tests to accommodate the desired change in the existing model. The big picture of regression is not limited to frequent release rather it is concerned with satisfaction after release. This satisfaction is dependent upon non breakable code. Regression testing and definition of done for user story are checked for every quadrant of the agile matrix of Lisa Crispin ‘s book “Agile testing”. Question is whether to use tools for specific type of testing or not. Except for quadrant Q3, in all the quadrants tools can be used and selection can be done on the type of testing in the improved agile quadrant matrix of figure 1.

For Q3, manual approach is used for usability, exploratory and scenario testing. User acceptance testing can be performed using Acceptance TDD which is xUNIT based, Fitnesse and through continuous integration. Scenario testing can be performed using UML use cases. Input is taken by team members through customers. From an agile point of view, these use case diagrams help to refine the problem space in more broader way so that every stakeholder is aware of the problem-solution pair. Functional Tests can be tested by Selenium like open source tools. Q1 quadrant is the core area while performing regression testing, as internal as well as external quality checks are performed by its team members. In Q1, TDD and refactoring plays a significant role. Refactoring is performed on the code when new feature is added, when defect in the existing code is to be removed and when code review is done by

team members. With the help of comprehensive testing of unit tests and acceptance tests suite, step wise refactoring/rewriting of the code is performed so as to have simple design in XP methodology of agile model. In this way, program code can be maintained for the long term. Simple design of XP helps in integration of the code and test suites. Simple design is nothing but rearrangement/restructuring of the statements of the program code by using object oriented principles so as to get rid of the bad design. Also, Class Responsibility Collaborator (CRC) card, which was invented in 1989 by Kent Back and Ward Cunningham, can be used for exploring objects or collaborator classes in the existing sprint deliverable r requirement. This exploration of collaborators or associations will help in finding the relationship among classes and further, for applying rules of refactoring to the bad design code to get the clean code having flexibility and scalability feature.

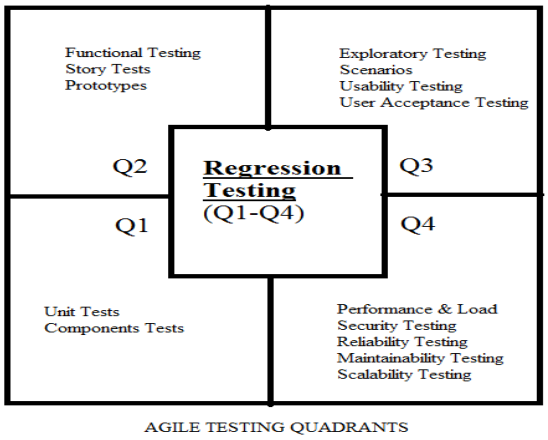

Figure 1. *Improved Agile Testing Quadrant Matrix*

The process flow diagram (refer figure 2) for the step wise conversion of the object oriented source code into improved code (rewriting or refactoring) is comprised of various steps such as source code, test suite of unit tests (UT) and acceptance tests (AT), regression testing and finally acceptable and release of rewritten code. Source code may be rewritten when change request is created for the existing module, when code review is performed by the pair programming individuals or reviewers and when new defect or problem of high severity is detected. Refactoring is applied on the original source code by applying object oriented principles so that there can be escape from the bad design. After doing refactoring of the source code, same test suite having unit tests and acceptance tests are applied to the new code until syntactic and semantic correctness is achieved. Also, behavior remains consistent by restructuring the statements of the original source code. This correctness check is precondition before the release. As agile is iterative and incremental, that's why regression testing is also incremental. Regression testing in this paper is limited to unit tests and acceptance tests but it is not only limited to these two tests rather it can include all types of testing which are covered under the improved agile testing quadrant matrix. It is an ongoing process in agile. The bulky size of test suite makes the testing task cumbersome. So, to release the deliverable on time, regression test

selection can be applied which is based upon optimal connection technique. The optimality of relationship of the user stories can be identified by using two parameters average path value and average path length. This technique will save time, resources and quality check on refactored code would be performed by retaining the original behavior of the source code. Regression testing in this proposed model is performed manually but to speed up the process it can be done by using different automated open source tools.

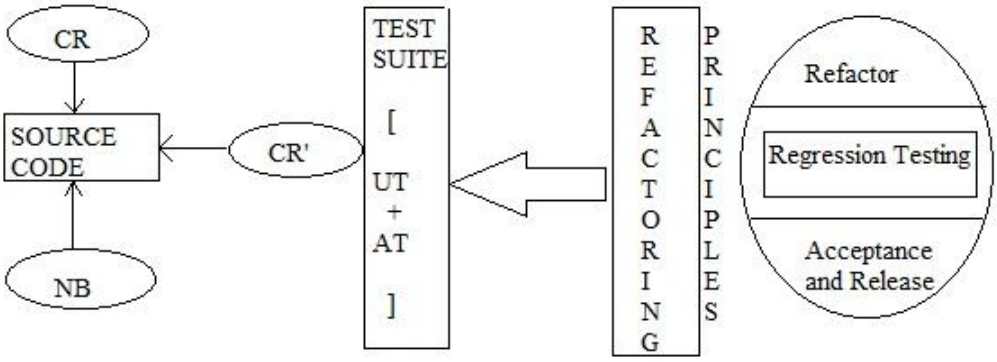

Figure 2. *Step Wise Source Code Refactoring*

4. AUTOMATED PLUGIN (XREFACTORY)

Xrefactory is a source understanding and refactoring plugin which can be seamlessly integrated into existing devel- opment environments and editors. Xrefactory is a professional development tool for C and C++ providing code completion, source browsing and refactoring. It is "a must have" for understanding legacy code. While Xrefactory for Java implements different refactorings, such as: rename class, package or symbol (includes variables, parameters, etc. renaming) add, delete or reorder parameters (of a function or method) extract method(take a piece of code and generate new method) expand or reduce names (expand names to FQT form, or reduce over qualified names) move field (between classes) move static method(between classes) move class (between files) pull up or push down field or method (in the class hierarchy) encapsulate field(generate getField and setField methods and replace all occurrences of the field by those methods). Turn virtual method to static turn static method to virtual.

5. CONCLUSIONS

The proposed model of refactoring which is based upon design principles and regression testing ensures that bad design is converted into consistent code having simple design that is the fundamental requirement in XP methodology of agile approach of software development. This model ensures that precondition of syntactic and semantic correctness is achieved after refactoring the code. For this, test suite comprising of unit tests and acceptance test is run iteratively so as to have the good design. Also, ways of refactorings are mentioned by way of xrefactory plugin.

Acknowledgements:
We would like to thank the reviewers for their useful suggestions that helped us to improve our work. We would also like to extend our gratitude to our parent institute who provided us with the needed research facilities.

6. REFERENCES

[1] A.J. Ko, H.H. Aung and B.A. Myers. Eliciting design requirements for maintenance-oriented IDEs: A detailed study of corrective and perfective maintenance tasks. *Proceedings of 27th ICSE*, pp. 126-135, 2005.

[2] Balaban, F. Tip and R. Fuhrer. Refactoring support for class library migration. *Proceedings of the 20th OOPSLA*, pp. 265-279, 2005.

[3] D. Dig and R. Johnson. The role of refactoring in API evolution.*Proceedings of ICSM*, 2005.

[4] J. Henkel and A. Diwan. CatchUp! Capturing and replaying refactoring to support API evolution. *Proceedings of the 27th ICSE*, pp. 274-283, 2005.

[5] K. Beck. *Extreme Programming Explained: Embrace Change.*Addison-Wesley, 1999.

[6] M. Fowler. *Refactoring: Improving the Design of Existing Code.* Addison-Wesley, 1999.

[7] W.F. Opdyke. *Refactoring Object-Oriented Frameworks.* Ph.D. Thesis, University of Illinois at Urbana-Champaign, 1992.

[8] Z. Xing and E. Stroulia. UMLDiff: An algorithm for object oriented design differencing. *Proceedings of the 20th International Conference on Automated Software Engineering*, 2005.

[9] Konig, H., Lowe, M., Peters, M., Schulz, Ch., A Formal Framework for Information System Refactorization, FHDW Hanover, Freundallee 15, D-30173 Hanover, 2006

[10] Luxoft UK Limited. FITpro – Acceptance Testing Solution. http://sourceforge.net/projects/fitpro (02.10.2008), 2008.

[11] R. Mugridge and W. Cunningham. Fit for Developing Software. Pearson Education, Inc., New Jersey, 2005.

[12] T. Widmer. Unleashing the Power of Refactoring. IBM Rational Research Lab Zurich, February 2007.

[13] L. Crispin and J. Gregory, Agile Testing: A Practical Guide for Testers and Agile Teams, ISBN-13: **978-0321534460** | Edition: **1**

[14] A. Arora and Naresh Chauhan, "A regression test selection technique by optimizing user stories in an agile environment", unpublished, 4th IEEE International Advance Computing Conference, Feb 21-22, 2014

[15] A. Arora and Naresh Chauhan," A Simplest Agile Life Cycle for All Stakeholders", 7th international conference on software engineering, 15th -17th Nov 2013, Pune, India

[16] W. Cunningham. Framework for Integrated Test – Fit. http://fit.c2.com/ (10.10.2008), 2008.