ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

IJARCCE

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 12, December 2016

Component Based Software Architecture Refinement and Refactoring Method into Extreme Programming

Mrs. Nagalambika Swamy[1], Dr. L. Manjunath Rao[2], Mr. Praveen K S[3]

Asst. Professor, Department of MCA, East West Institute of Technology, Karnataka, India[1, 3]

Professor and Head, Department of MCA, Dr. Ambedkar Institute of Technology, Karnataka, India[2]

Abstract: Extreme programming is an agile methodology for software development that performs very well with changing requirements. XP is one of the most commonly used methods among other agile methods. However, it is implemented sequentially on all activities Moreover; classical XP suffers from an architectural design. Therefore, there is a need for a framework that integrates the strengths of component based architecture refinement reusability into the Extreme Programming Methodology. Which gives a clear vision about a current architectural design requirement without any additional features that are not yet needed? And constantly redesigning through refinement and refactoring concept. The design is simple and loosely coupled as possible, thus making future modifications easier, and achieving the XP values i.e. simplicity and feedback. This will result in reusability of component architecture and to reduce the development effort, time and provide quality software.

Keywords: Refinement, Refactoring, Component, architecture reuse, Agile Software Development, Extreme Programming.

I. INTRODUCTION

Extreme Programming (XP) is an iterative and incremental agile software development method. That aims to develop software in environments of unclear and changing requirements. The XP focused highly around customer satisfaction and involvement, incremental delivery of software and stakeholders' collaboration and cooperation. These principles help projects with changing requirements to succeed. In XP, programmers develop the system in pairs. Code is well tested and reviewed. Only the current functional requirements are focused, without any additional features that are not yet needed extreme programming demonstrated dynamism through four values:

- Simplicity − achieved by a constant focus on minimalist solutions.
- Communication − continual communication with the customer and within the team
- Feedback − rapid feedback through mechanisms such as unit and functional testing
- Courage − the courage to deal with problems proactively

Although developers might use many different XP practices, the method typically consists of 12 basic elements:

- Planning game − quickly determine the next release's scope, combining business priorities and technical estimates. The customer decides scope, priority, and dates from a business perspective, whereas technical people estimate and track progress.
- Small releases − Put a simple system into production quickly. Release new versions on a very short (two-week) cycle.
- Metaphor − Guide all development with a simple, shared story of how the overall system works.
- Simple design: Design as simply as possible at any given moment.
- Testing − Developers continually write unit tests that must run flawlessly; customers write tests to demonstrate that functions are finished. "Test, then code" means that a failed test case is an entry criterion for writing code.
- Refactoring − Restructure the system without changing its behavior to remove duplication, improve communication, simplify, or add flexibility.
- Pair programming − All production code is written by two programmers at one machine.
- Collective ownership − anyone can improve any system code anywhere at any time.
- Continuous integration − Integrate and build the system many times a day (every time a task is finished). Continual regression testing prevents functionality regressions when requirements change.
- On-site customer − Have an actual user on the team full-time to answer questions.

II. LITERATURE-REVIEW

Software architecture research domain is rapidly evolving as the systems increase in size and complexity [3]. The

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319 5940
International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 12, December 2016

architecture of a software system can have a major impact on system efficiency, maintainability, and evolvability.

System development by refinement is a formal model-driven development process. Re-finement allows us to ensure that a refined, i.e., more elaborated, model retains all the essential properties of its abstract counterpart. Since refinement is transitive, the model-driven refinement-based development process enables development of systems correct-by construction [8].

The precise definition of refinement depends on the chosen modelling framework and hence might have different semantics and the degree of rigor. The foundations of formal reasoning about correctness and stepwise development by refinement were established by Dijkstra [1-2], and then further developed by R.J.R .Back [5]. The component based architecture refinement framework, component is the most basic element of software, first of all the system as a high level of abstract component, describe the specification of component in accordance with its functional requirements, and then decompose it to generate a number components including data, ports, behaviours' of components[4].

Extreme programming is a methodology for software system development that focuses on high customer integration, extensive testing, code-centered development and documentation, refactoring and paired programming. In this approach every 3-4 weeks, a new fully functional release is delivered and reviewed by the customer. The specifications for each release are captured incrementally using use case scenarios [6].

Stepwise refinement is a top-down design strategy used for decomposing a system from a high level of abstraction into a more detailed level (lower level) of abstraction. At the highest level of abstraction, function information is defined conceptually without providing any information about the internal workings of the function or internal structure of the data. As we proceed towards the lower levels of abstraction, more and more details are available [5].

III.PROPOSED WORK

A. Problem Identification

Despite the benefits offered by XP, several drawbacks are noted:

- XP doesn't work with distributed development environment. The reason is that the practices and activities of XP require high collaboration, involvement, and face-face meeting and customer to be with the team. Due to the high agility of XP and simplicity of its design
- The system is developed very quickly without taking reusability of newly added components into account.
- XP doesn't take the advantage of component based architecture into account.
- XP is not suitable with outsourced team. This is because the XP needs highly competent members in team. These members need to collaborate, trust, respect and be self organized. Such skills are hard to find in outsourced team members who work just for the project that is assign to them.
- XP suffers from weak documentation
- Lack of overall design for the system.

B. Proposed Solution

The frameworks aim to keep architecture design be constantly redesigning through refinement and refactoring. The design is simple and loosely coupled as possible, thus making future modifications easier, further automated testing tries to ensure that modifications and refactoring do not create faults in the existing code.

We combine the salient aspects of software reuse and agile development and present an integrated approach to promote component architecture reuse. This approach is aimed at embedding architecture reuse as a standard practice within Extreme Programming.

Our proposed reuse process model is shown in Figure 1.

It consists of the following steps:
Step 1 - Component search and retrieval
Step 2 - Identify components to extend and refine
Step 3 - Generate target components, and
Step 4 - Repository management.
Each of these steps is briefly described below.

Step 1 - Search and Retrieval**:** The objective of this step is to search and retrieve relevant components from the reuse repository that can meet the current requirements of the system. Components that implement the methods that support the functionalities desired in the target system are searched. The output of this step is the initial set of potential components that may be relevant to the system requirements [6].

Step 2 - Identify Components to Extend and Refinement: The initial set of components is further examined to select the most appropriate components. In this step, we focus on identifying which parts of the component are directly relevant and which parts need to be eliminated. Components may have to be extended with additional functionalities to meet the requirements. The components that directly satisfy a requirement may also have problems as they may not have been designed properly be highly structured or efficient. Based on the components' goodness of fit, they are flagged as candidates for customization, extension, or refactoring.

Step 3 - Generate Target Components**:** This step focuses on creating the components that are suitable for inclusion in the application that is currently under development. These components are created either through extension and customization, or through refactoring, or both.

Step 3a - Extend/Customize Component: Components that partially support a requirement may have to be extended or customized by adding new properties or customizing an existing property.

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319 5940
International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 12, December 2016

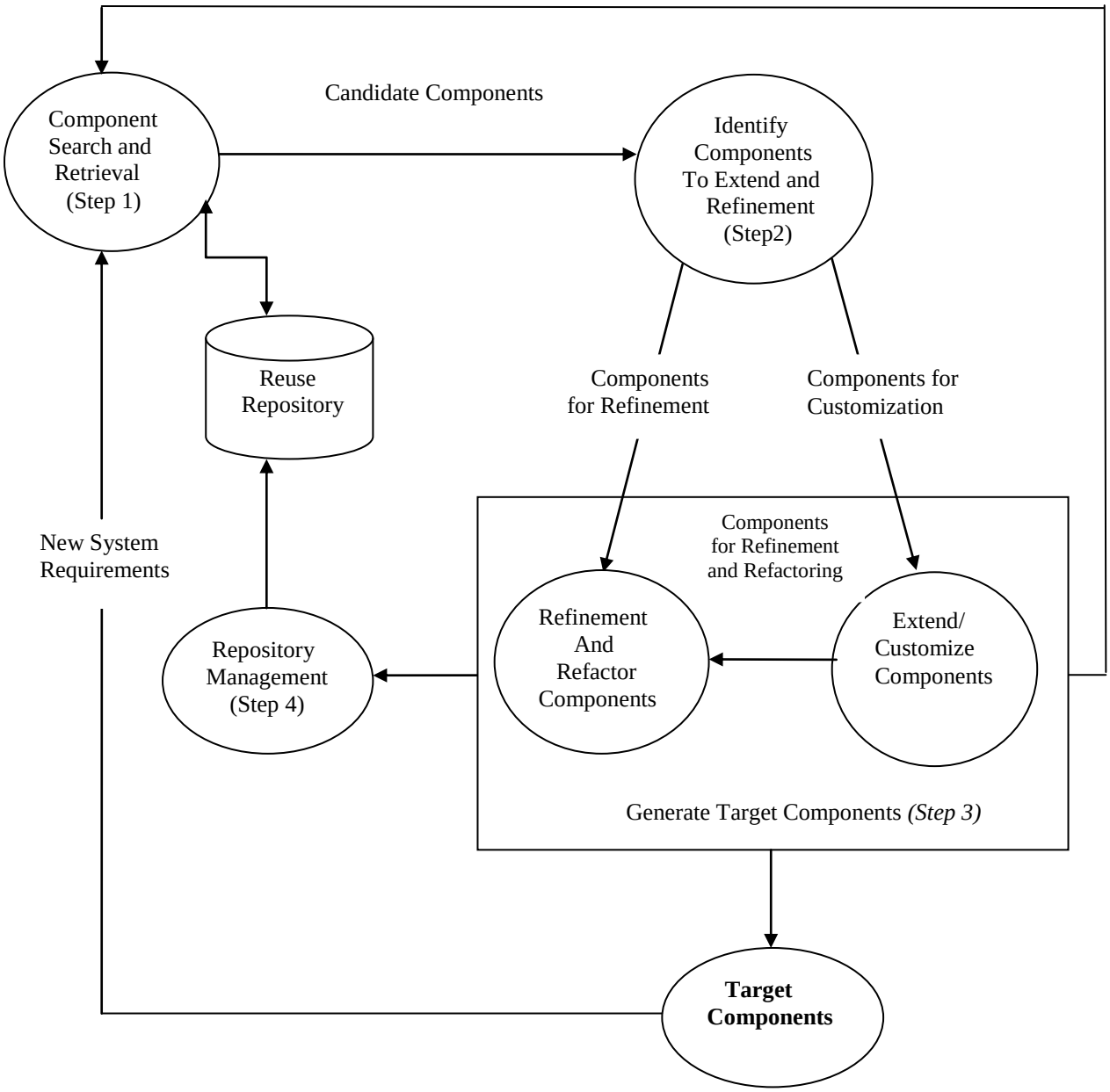

Fig. 1 Component Based Architecture Refinement and Refactoring Reusability Model for Extreme Programming cycle

The extension and customization of components changes their behaviour and hence needs to be tested. There is a feedback mechanism between this step and the initial step of component identification. The modified components may also be refinement and refactored to make them more efficient and be suitable for the application that is being developed.

Step 3.1 - Refinement Component: Stepwise refinement is a top-down design strategy used for decomposing a system from a high level of abstraction into a more detailed level (lower level) of abstraction. At the highest level of abstraction, function or information is defined conceptually without providing any information about the internal workings of the function or internal structure of the data. As we proceed towards the lower levels of abstraction, more and more details are available. Software designers start the stepwise refinement process by creating a sequence of compositions for the system being designed. Each composition is more detailed than the previous one and contains more components and interactions. The earlier compositions represent the significant interactions within the system, while the later compositions show in detail how these interactions are achieved.

Step 3.2 - Refactor Component: Refactoring is an important design activity that reduces the complexity of module design keeping its behaviour or function unchanged. Refactoring can be defined as a process of modifying a software system to improve the internal structure of design without changing its external behaviour. During the refactoring process, the existing

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319 5940
International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 5, Issue 12, December 2016

design is checked for any type of flaws like redundancy, poorly constructed algorithms and data structures, etc., in order to improve the design. For example, a design model might yield a component which exhibits low cohesion (like a component performs four functions that have a limited relationship with one another). Software designers may decide to refractor the component into four different components, each exhibiting high cohesion. This leads to easier integration, testing, and maintenance of the software components.

Step 4 - Repository Management: All the components created from the previous step along with their relationships are stored in a repository for future use. Consistency checking and repository management is an essential aspect of this approach. The component searching process should take into account these relationships. The retrieved components are most useful if they directly meet the requirements and are consistent with each other. Storing the new components in the repository takes into account feature.

IV. METHODOLOGY

- Unified modeling Language
- IBM Rational Rose Software

V. FINDINGS

- Extreme programming can be used as both code and design centric development.
- Reusability of Component design reduces the development effort, time, and cost and provides quality software.
- The design is simple and loosely coupled as possible, thus making future modifications easier, and achieving the XP values i.e. simplicity.
- Framework gives a clear vision about a current architectural design requirement without any additional features that are not yet needed. This helps in rapid development.

VI. CONCLUSION

This paper gives a clear vision about a current architectural design requirement without any additional features that are not yet needed. And constantly redesigning through refinement and refactoring concept. The design is simple and loosely coupled as possible, thus making future modifications easier, and achieving the XP values i.e. simplicity and feedback. It helps to reduce, the development effort, time and provide quality software.
This framework combine the features of two important XP practices that is
1. Simple design and
2. Refactoring

Concept into component based architecture refinement framework.

ACKNOWLEDGEMENT

I am extremely thankful to express my sincere gratitude to my Parents, Husband, Guide and my colleagues for their kind co-operation and for providing valuable suggestion and constant encouragement for the improvement and successful comp

REFERENCES

[1] E. W. DIJKSTRA, A constructive approach to the problem of program correctness, Bit 8 (1968), pp. 174-86.
[2] E.W.DIJKSTRA, "A Discipline of Programming," Prentice-Hall, Englewood Cliffs, N.I, pp. 1976.
[3] Aleti, A., Buhnova, B., Grunske, L., Koziolek, A., & Meedeniya, I., Software architecture optimization methods: A systematic literature review. IEEE Transactions on Software Engineering, 39(5), 2013, pp. 658-683.
[4] Bingzhi Gao; Xiaojuan Ban; Qiang Lv; Xiaoli Li, A component-based method for software architecture refinement, International conference on Intelligent Control and Information Processing, pp. 574-578
[5] R.J.R Back, On Correct Refinement of Programs, Journal of computer and system science 23, pp. 49-68, 1981.
[6] Elmuntasir Abdullah, El-Tigani, B. Abdelsatir, Extreme programming applied in a large-scale distributed system, International Conference on Computing, Electrical and Electronics Engineering (ICCEEE), ISBN:978-1-4673-6231-3,pp.442-446, 2013.
[7] M. Moriconi, X. Qian, R. A. Riemenschneider, Correct Architecture Refinement, IEEE Transactions on Software Engineering, ISSN :0098-5589, pp. 356-372, 1995.
[8] Alexei Iliasov, Elena Troubitsyna, Linas Laibinis, Alexander Romanovsky, Patterns for Refinement Automation, ISBN: 978-3-642-17070-6, pp. 70-88, 2010.
[9] Gerald DeHondt et al., CODE REUSE AS A PRACTICE WITHIN EXTREME PROGRAMMING
[10] https://www.tutorialspoint.com/software_architecturedesign/component_based_architecture.htm

BIOGRAPHY

Mrs. Nagalambika Swamy is working as Asst. Professor, Department of MCA, EWIT, Bangalore. She has worked in various Software Industry and Educational Institutions. She has total of 10 years of experience in Software Industry and Educational field. India

Dr. L. Manjunatha Rao is working as Professor and Head, Department of MCA, Dr.AIT, Bangalore. He has awarded Ph.D from Vinayaka Mission University, Tamil Nadu and obtained Ph.D degree from SV University, Tirupati, Andrapradesh. He has published research papers in both national and international Journals and has authored 2 textbooks.

Mr. Praveen K. S is working as Asst. Professor, Department of MCA, EWIT, Bangalore. He has total of 7 years of experience in Educational field. India