CODE REUSE AS A PRACTICE WITHIN EXTREME PROGRAMMING

Gerald DeHondt\textsuperscript{a, b}, Vijayan Sugumaran\textsuperscript{b}

\textsuperscript{a} Department of Management and Information Systems, College of Business Administration
Kent State University, Kent, OH 44242
gdehondt@kent.edu, dehondt@oakland.edu, (330) 672-1144

\textsuperscript{b} Department of Decision and Information Sciences, School of Business Administration
Oakland University, Rochester, MI 48309
sugumara@oakland.edu, (248) 370-2831

ABSTRACT

Reuse based software development focuses on improving productivity. Agile techniques also seek to improve the traditional plan-based development methodologies. Both approaches bring value to the software development process and should be combined. This paper discusses a framework that integrates the strengths of code reuse into the Extreme Programming methodology. This will result in a more effective method for software development.

Keywords: Code Reuse, Agile Software Development, Extreme Programming

INTRODUCTION

Code reuse has been practiced for more than thirty years in software development with varying degrees of success [6]. Individual developers have managed to cut and paste previously written code into new programs when appropriate. This is referred to as opportunistic or ad hoc reuse and it works fine in a limited way for individual programmers or small groups. However, it does not scale up across the organization to provide systematic reuse. Hence, considerable research and effort has been spent on developing approaches, methodologies, tools and techniques for facilitating and improving systematic software reuse. If systematic reuse is practiced effectively within an organization, it stands to gain much competitive advantage in terms of reducing time-to-market, developing quality products, and being able to adapt to changes in the market place. While some success stories have been reported in the literature, efficient and seamless software reuse has proven to be elusive even for the most successful software development companies [5].

Software reuse is a valuable method for increasing the efficiency and effectiveness of an organization’s software development efforts [8] [17]. Current implementations of this process have encountered some challenges such as lack of incentives, lack of available resources, suitable component identification, and necessary tools for customization and validation among other items. In spite of this, appropriate component reuse can integrate previously implemented software into current development projects serving to propagate validated code within the application infrastructure [16].

The purpose of this paper is to propose code reuse, properly applied and implemented, as an extension to the existing practices of Extreme Programming. In order to be successful, reuse components must be developed utilizing the practices and values embodied within the XP
methodology. Development of “XP-Compliant” reuse components will allow these components to be successfully integrated into future development projects.

**LITERATURE REVIEW**

**Extreme Programming**

Extreme Programming (XP) is an agile development methodology emphasizing rapid and frequent feedback to the end users, unit testing, and continuous code reviews. By focusing on rapid iterations of simpler code, XP seeks to identify and resolve potential pitfalls in the development process early, leading to projects that remain focused on the ultimate goal – timely delivery of a well-designed and tested system that meets user requirements. It is also described in terms of the values that support it: communication, feedback, simplicity, courage, and respect [1]. This methodology works by bringing the whole team together in the presence of simple practices, with enough feedback to enable the team to see where they are and tune the practices to their unique situation.

XP works by breaking down a project into sub-projects, each including planning, development, integration, testing and delivery [10]. Features are implemented iteratively during each development cycle with joint decision making occurring between the customer and the rest of the development team. Agile software-development methods use human- and communication-oriented rules in conjunction with light, but sufficient, rules of project procedures and behavior [4]. They rely on planning, with the understanding that everything is uncertain, to guide the rapid development of flexible systems of high value [9] [10]. XP seeks to implement the simplest design that will satisfy current user requirements [12] without attempting to anticipate future design or user requirements.

Nerur and Balljepally [13] state that agile methods are people-centric, recognizing the value that competent people and their relationships bring to software development. In addition, agile methods focus on providing high customer satisfaction through three principles: quick delivery of quality software; active participation of concerned stakeholders; and creating and leveraging change [9].

Agile development is also characterized by social inquiry in which extensive collaboration and communication provide the basis for collective action [3] [4] [10]. Reuse of software components provides an exceptional form of collaboration, going beyond the boundaries of the current project to propagate and cultivate superior methods and software throughout the organization.

Organizations undertaking agile methodologies must invest in tools that support and facilitate rapid iterative development and versioning/configuration management [5]. One way to do this is through investment in a suitable reuse strategy supporting agile development. Even with the most developed repository in place, organizational measures and strategies must be implemented to support the process, including training staff to make the most effective use of this corporate asset [5].
Software Reuse

Software reuse has been an important area of research within software engineering and several approaches have been discussed in the literature to improve reuse that utilize domain modeling, component based development, software product line, design patterns, etc. These approaches primarily attempt to realize the benefits of reuse for object-oriented conceptual design through the creation of tools to facilitate design and construction of new systems with reuse [18]. Higher-level design fragments and models are also being developed [15] that facilitate the easy use of preexisting artifacts. Clayton et al. [2] suggest that it would be useful to have a set of previously solved design problems, in the form of domain specific software architectures that could serve as templates for future designs.

In ad hoc or opportunistic reuse, the scope of the reuse effort is narrow in that a single developer might try to reuse artifacts developed by him or her in prior projects. The developer has limited resources and the number of reusable artifacts is minimal. Also, developers do not share their reusable assets. Systematic or organizational reuse on the other hand is larger in scope and reuse is practiced or mandated throughout the organization. Reusable assets are stored and maintained in central repositories and everyone within the organization can access them. Considerable effort and resources are spent in creating reusable assets and making them available to all the projects.

Systematic software reuse provides a promising means to reduce development cycle time, development cost, and improve software quality. Particularly, if reuse is materialized in the early stages of systems development by reusing requirement specifications, designs, and other higher level artifacts, the project team has a tremendous opportunity to drastically cut the overall development time and effort. However, many factors conspire to make systematic software reuse a non-trivial task. For example, contextual and behavioral issues such as management support and the software developer’s interest are seen to be more important than technical issues in determining the success or failure of software reuse programs [11].

In order for reuse to be successful, the tools and techniques used should be flexible, agile and be able to support multiple systems development life cycle models. In particular, since agile development methodologies are gaining prominence, it is worthwhile to incorporate reuse practices into the context of agile development. We propose a framework that specifically does that, which is described in the following section.

**PROPOSED FRAMEWORK**

The reuse literature discusses many approaches and methodologies for institutionalizing software reuse, however, none of them have shown to be highly successful. Practitioners of agile software development methodologies suggest that in order for software reuse to be successful, particularly in the agile development environment, the essential aspects of the individual or ad

---

1 The authors discussed this idea and received input from Chet Hendrickson and Ron Jeffries, two well-known Extreme Programming practitioners. They were original team members of the seminal XP Project at Chrysler Corporation in 1996, and have since provided consulting and training services to Fortune 500 companies in the appropriate use and implementation of the Extreme Programming methodology and spoken at numerous academic and practitioner conferences on this topic.
hoc reuse must be abstracted out and create the necessary infrastructure and mechanisms that will implement these aspects to facilitate reuse on a larger scale. They aptly point out that software reuse is more of a social process as opposed to a technical process. It is our contention that the tools and techniques developed to improve software reuse should incorporate practices from agile software development.

We combine the salient aspects of software reuse and agile development and present an integrated approach to promote reuse. This approach is aimed at embedding code reuse as a standard practice within Extreme Programming. Our proposed reuse process model is shown in Figure 1. It consists of the following steps: 1) component search and retrieval, 2) identify components to extend and refactor, 3) generate target components, and 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 requirements of the new system. This step can integrate domain knowledge, component standards, and architecture knowledge. Components that implement the procedures/methods that support the functionalities desired in the target system are searched. To identify the most relevant components, the operations of a component are compared to the desired functionalities expressed in the user’s requirements using the closeness measure proposed by Girardi et al [7]. The output of this step is the initial set of potential components that may be relevant to the system requirements.

![Figure 1. Reuse Practice Process Model for Extreme Programming](image)

**Step 2 - Identify Components to Extend and Refactor:** 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. The extension and customization of components changes their behavior 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 refactored to make them more efficient and be suitable for the application that is being developed.

**Step 3b - Refactor Component:** Several refactoring techniques exist [19] and one or more of these techniques can be applied to generate components. As the refactoring step gets executed over a period of time, one can detect similarities and patterns in what and how components get customized and how this knowledge could be used in component identification. Hence, this step has a feedback mechanism. Another important aspect is that the behavior of the components should not be changed. This step also ensures that the pre-conditions, post-conditions, invariants and temporal constraints are preserved.

**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 control and version management. The components are tested thoroughly and the test cases are updated for the new version. Similarly, the documentation is also updated to reflect the new structure and behavior of the component.

**DISCUSSION**

The method being proposed seeks to extend the practices of Extreme Programming by adding software reuse. Already inherent within Extreme Programming are practices that integrate well with reuse including refactoring, test driven development, and collective ownership. Additionally, reuse integrates well with the communication value of XP.

The proposed model integrates code reuse into the XP practice of refactoring. It is also believed that in their reuse efforts, developers will integrate XP methods into their development efforts. This would include full documentation of the software and fully specified test cases and test results for each component. It is envisioned that by reusing a component, developers will have access to this information about the component to integrate into their development efforts. Disseminating the information about the component, how it works, and anticipated results will place developers much further along the learning curve allowing more efficient software development. Access to fully specified test cases and test results will help validate the
functionality of the component and add efficiency to the software development process and reuse efforts.

Software reuse also embodies collective code ownership, another XP practice. Making software components fully available to the organization, and implementing methods and techniques to speed identification of appropriate components, represent collective ownership of the code base. Any developer in the organization is encouraged to reuse and improve any component that meets their needs. These efforts provide an evolving and improving code base from which to develop future software and satisfy customer requirements.

As mentioned earlier, software reuse is a form of communication among software developers, providing solutions to earlier problems. As these problems are re-encountered, reusing previously effective solutions helps accelerate the development cycle. One of the challenges to reusing components is ensuring that they are easily identifiable, suit the task at hand, and can be easily refactored or extended to improve performance or more appropriately meet current needs.

**CONCLUSION**

This paper continues work on extending and improving Extreme Programming by proposing an additional practice in the overall framework. As an Agile method, Extreme Programming is an evolving method, continuing to grow and improve over time. It is believed by the authors that integrating software reuse into Extreme Programming will help this methodology continue to grow and evolve to meet the needs of developers and customers. The authors also provide one cautionary note with regard to the integration of these two items. Code Reuse efforts must be performed and managed in a way that will work in an XP environment. Software to be reused in an Extreme Programming environment must already adhere to and have been developed according to the practices and values of XP. Reusing non-“XP-compliant” software in an Extreme Programming environment will not work. It is believed that the Extreme Programming methodology is a superior method of developing software comprised of best practices from previous development efforts. Properly applied, software reuse will integrate well with this approach as it touches on a number of the practices and values already inherent within Extreme Programming.

As an extension to existing practices, further work needs to be performed in validating software reuse as it applies to Extreme Programming, in addition to other Agile Methods. On the surface, it appears as it would naturally integrate into the Extreme Programming Framework, however much work is required to validate this fact and its applicability to XP. Future validation efforts would include a) identification and application of this framework to large scale projects to study its effectiveness, b) review of this approach in practical situations to confirm its validity or provide suggestions for refinement, and c) enhancement of this practice as required for improved performance.

**REFERENCES**

Full paper and references available upon request from the authors.