# A Practical Application of XP

**Kevan Dunsmore**
Trilogy®
6034 West Courtyard Drive
Austin
Texas 78730 USA
+1 512 532 5903
kevan.dunsmore@trilogy.com

**Charles Wiemann**
Trilogy®
6034 West Courtyard Drive
Austin
Texas 78730 USA
+1 512 532 5714
charles.wiemann@trilogy.com

**Gabriel Wolosin**
Trilogy®
6034 West Courtyard Drive
Austin
Texas 78730 USA
+1 512 532 5221
gabriel.wolosin@trilogy.com

**ABSTRACT**
This paper outlines the conclusions drawn from the Trilogy® project team's experience from an on-going consulting engagement employing Trilogy's Fast Cycle Time (FCT) methodology.

Some XP [1] practices are difficult to instigate in a "time and materials" consulting engagement. This document provides alternatives to these practices that preserve the spirit of XP. We describe requirements gathering, user interface development and general development practices that we have found to work well.

Through our experience on this project, we have gained the following insights: It is possible to gather requirements effectively without having the entire project team located at the customer site. A dedicated Human Computer Interaction (HCI) team helps boost User Interface (UI) quality and reduces cost. Three-week cycles provided the best balance between release administration and system development. Finally, employing a full-time Quality Engineer (QE) ensures the quality level that required for frequent software releases.

**Keywords**

XP, extreme programming, FCT, Fast Cycle Time, HCI, requirements, database, DB, software development, quality.

## 1 INTRODUCTION

Trilogy® [13] was awarded the contract to design and implement a new call center application for one of the largest retail catalog merchants in the United States. The merchant started with a strategic vision of incorporating customer intelligence into every step of the ordering process to provide the optimal customer experience across all channels. Trilogy's role was to assist in the business value modeling, scope the functional requirements, develop the functionality and user interface and implement and test the new functionality. Some examples of the types of functionality include search techniques for products, easier side-by-side product comparisons and new pricing techniques that are simpler for customers and more profitable for the merchant.

All Trilogy® projects are managed via the FCT methodology. FCT is a lightweight methodology similar to eXtreme Programming (XP). It promotes delivery of business-aligned solutions by employing multiple development and delivery cycles of two to four weeks in duration. These cycles are analogous to XP's iterations. FCT encourages continuous feedback and constant business prioritization of requirements to allow customers to realize Return On Investment (ROI) as quickly as possible. FCT also offers the added benefit of tracking specific professional service costs to specific functionality. This allows both Trilogy® and our customers to evaluate ROI for specific functionality as opposed to an overall project.

## 2 REQUIREMENTS GATHERING WITHOUT AN ONSITE CUSTOMER

We have found that it is possible to gather requirements successfully without an onsite customer with a small requirements team stationed at the customer site. Communication between the parties involved is vital. We have found that the requirements gathering process works best when it is sequenced at least one cycle ahead of development.

The concept of an "onsite customer" is valuable but can often be impractical. Clients are often unwilling to spare key individuals for remote efforts and the cost of prolonged travel can have severe impact on the budget of a project. In order to address these issues, a small requirements team works at the client's base of operations. This practice requires good communication between the requirements team and the development team. The requirements team has to be able to respond to questions quickly and the development team has to provide prompt feedback on the generated requirements.

On our project, requirements are gathered in cycles with the requirements team working one or more cycles ahead of the development team. This ensures that there are requirements available and ready to be built at the beginning of each cycle.

Prioritized high-level functional requirements provide a "road map" from which work is scheduled. In each session, a "use case" [4] is produced, similar in form to a "user story" [1]. These use cases are employed to provide the context for gathering functional and technical requirements. During development, the assigned programmer determines the details of each use case in conjunction with a member of the requirements team and a designated customer domain expert.

FCT allows the customer to reprioritize and change requirements to meet varying needs. In our experience, the reprioritization of requirements usually has little impact on work under development or on the use cases already in the one-cycle-ahead "pipeline." When development is affected, it is affected in one of two ways. In the first situation, a change affects previously completed work. In this instance, new requirements are gathered and fed into the pipeline with the appropriate customer-designated priority. The development team estimates the time required for the change and the customer decides if the new feature is financially viable. In the second situation, a change affects requirements in the pipeline. In this scenario, it is possible that development "stalls" at the beginning of the next cycle Typically, this "stall" happens at the beginning of a project, when there are relatively few use cases. In this situation, the impact of the missing cycle is communicated to the client. The customer must then decide if the change is worth the schedule slippage.

On this project, the development cycle stalled on two occasions. In an attempt to create a solution that could be used across several channels, our client included representatives from several business areas in the use case development process. On both occasions, competing business needs prevented the group from reaching consensus. This situation occurred early in the project, before a backlog of use cases had been produced. The result was a work stoppage. Strong executive sponsorship was required to break these deadlocks and prevent development cycle stalls.

In conclusion, it is possible to gather requirements without an onsite customer successfully. By stationing a specialized requirements team at the customer site, similar benefits can be realized. This strategy requires excellent communication between the development team, the requirements team and the customer. The process works best if requirements are gathered at least one cycle ahead of development and a strong executive sponsor serves as the arbitrator of functionality disputes.

## 3 USER INTERFACE DEVELOPMENT

Under FCT, the development of a cohesive, intuitive user interface poses some difficult questions. As with XP, system functionality is often developed in "verticals"; a particular application function is known in great detail but there is often less application knowledge in breadth, particularly at the beginning of development. Since this breadth of knowledge on the part of the designer is key to the production of a cohesive interface, this causes difficulty.

Our approach has been to delay the development of a cohesive interface and instead concentrate on developing intuitive interfaces for each functional vertical. There are often obvious areas of commonality between the interfaces of each vertical that can be refactored to increase cohesiveness in the short term. For example, ensuring that buttons labeled with the same text perform the same function and that dialog layout is consistent. However, the major UI design work has been postponed until there is sufficient application knowledge and, more importantly, there is a pressing business need for UI cohesiveness.

On our project, we have a separate HCI team that works closely with the requirements team. As with requirements, HCI works one or more cycles ahead of development. Our HCI team consists of two people. One person works on determining common system usage patterns and works closely with the requirements team. The second person designs the UI and works closely with the system developers.

The first step in designing the UI is to build wire frame models using commercially available software. Our project's choice of software is Microsoft's PowerPoint. PowerPoint facilitates creating graphics quickly and provides the ability to build "slides" which detail the progression of an interaction. The PowerPoint presentations are used to guide the client through an interaction before any code is written. The development of each feature's UI takes several iterations. Each iteration includes feedback from the client and the development team. At the end of the HCI cycle, the slides represent a UI that is acceptable to the client and is technically feasible to build.

Postponing as many HCI decisions regarding interface cohesiveness and focusing on intuitive interfaces for functional verticals allows a UI of fair quality to be developed. This approach means that a UI refactoring cycle at a later point in the project is likely. The inclusion of an experienced HCI team has reduced a large amount of UI rework on the part of the system developers. The reduction in rework depends on the HCI cycles taking place at least one cycle ahead of development.

## 4 DEVELOPMENT

In order to achieve the desired results, the development team must work closely with the requirements, HCI and customer teams. Estimates are developed during group sessions with established ground rules to ensure efficiency and accuracy. A modified Classes

Responsibilities and Collaborations [2] (CRC) process is used for design. The CRC cards are modified to communicate which aspects of the system design map to particular business requirements. The difficulty of persuading a client of the benefits of pair programming was overcome by implementing a two-stage code review policy.

After experimenting with several cycle lengths, the development process was organized into 3-week cycles. The first week is reserved for addressing requirements questions and fixing any problems from the previous cycle discovered during end-to-end system and load testing. The second and third weeks are reserved for development.

In the first week of the cycle, developers review the target use cases and compile a list of questions jointly. The onsite requirements team and the client work to resolve these questions as a team. Often, some of the questions are answered over the phone during the meeting. When all questions are answered, the development team breaks the use cases into tasks and estimates each feature collectively. Some basic rules apply to the estimation process. No task can be estimated at less than 0.5 days. Estimates must include the time required for design and testing. When estimation is complete, each developer volunteers to implement one or more tasks. No developer may sign up for more days of development effort than there are working days in weeks two and three, ensuring that the team doesn't over-commit.

The first week of a cycle also provides an opportunity for developers to address any problems found in end-to-end system testing. See section 5 for an explanation of end-to-end system testing.

The second and third weeks are reserved for writing code for both system features and tests. Tests are written and executed automatically by an internal tool similar to JUnit [5] and optimized for execution within the Trilogy® product suite.

The first development cycles were two weeks in duration. The amount of time spent creating and running end-to-end tests and packaging the application for delivery in proportion to the amount of time spend developing the system was impractical and cost prohibitive. The 3-week cycle strikes a good balance between the time spent supporting frequent deployments and developing the system features.

System design is performed with a modified CRC process. At least two developers attend each session. CRC cards are used to describe the classes and the relationships required for each feature. On each card, collaborators and responsibilities are assigned a number corresponding to a use case task number. When the design is complete, the card contents are added to a document called the "Cycle Design Report" (CDR), which is sent to the client. The CDR allows client's technical staff to follow the system design and trace each major class to a particular business requirement, ensuring accountability. Each design element must correspond to a business requirement.

In a "time and expenses" engagement, pair programming, a staple of XP, is controversial. We have found it difficult to persuade clients to pay two people to write the same piece of code, regardless of the proven benefits of the practice. In order to preserve the spirit of pair programming, we have implemented a two-stage code review process at different stages of development.

Code reviews take place at two levels. The first level, a "code read", is used whenever a developer checks code into the source control system. Before code can be checked in, the developer will request another team member to review the code. The developer will explain the changes made and, using a differencing utility, show the changes to the previous versions of each source file. The reader examines the code to detect deviations from the project coding standards, obviously inefficient algorithms, re-factoring opportunities and documentation improvement opportunities. A code read can varies in length from 5 to 60 minutes. Code reads are most effective when a small number of files are being submitted.

Formal code reviews are the next level in the process. The entire development team reviews the previous cycle's code before the next cycle begins. The development team collectively determines whether the code base implements the feature set correctly and to the necessary level of efficiency. The team checks the code for design efficiency, correctness, thread safety, etc. A list of improvement suggestions is compiled and implemented after the review. A considerable investment of time, from a few hours to a day, is required for this process.

Although this two-stage code review process is not as comprehensive as pair programming, we have found it to be an acceptable compromise between the cost perceived by the client and code integrity.

In summary, 3-week cycles provide an ideal balance between the administration costs of frequent software releases and development time. The development team must work closely with requirements, HCI and customer teams. Group estimation sessions with established ground rules improve accuracy. Finally, client concerns with the cost of pair programming can be dispelled by a two-stage code review policy that approximates the benefits of pair programming.

## 5 ENSURING QUALITY RELEASES

Staffing the project with a full-time Quality Engineer (QE) has provided many project benefits. The QE relieves much of the administrative burden of the quality process from the rest of the project team. Several best practices have been identified to ensure acceptable quality. Well-defined quality metrics are required and the QE must have the authority to block releases if these metrics are not met. An appropriate set of software tools is also essential to the quality process. Software delivery is aided by an automatic build and test system that ensures the readiness of application for release.

Each developer is responsible for the quality of the code produced. The primary quality assurance mechanism is the creation of Application Programming Interface (API) tests that can be executed automatically. The Quality Engineer (QE), however, is ultimately responsible for the quality of the overall system release. The QE is responsible for creating and running end-to-end system tests and performance tests. End-to-end system testing is accomplished by using Segue's [11] SILK [12]. SILK scripts are created to simulate user interaction by executing use cases. Over time, a regression test suite is built. The SILK scripts are also used for system stress testing.

NuMega's [7] TrueCoverage [14] is employed to determine "code coverage," the percentage of delivered code exercised by the API tests written by the developers and the end-to-end SILK tests. The project targets of 80% function coverage and 70% line coverage are enforced by the QE. Failure to meet these targets prevents software release

The QE works within the development cycle timeframes. In the first week, the QE runs system tests and collects results. Issues are prioritized, communicated to the development team, and fixed during that week. Code delivery for the previous cycle is scheduled at the end of the first week of the new cycle. The second week is reserved for collection of issues raised by the client's User Acceptance Test (UAT) team. The third week is divided between UAT issues and the creation of SILK tests.

Although the combination of the end-to-end tests and system API tests reduces the number of issues discovered during UAT, the process isn't perfect. Rational's [10] ClearQuest [3] is used to manage both UAT and internal issues. The QE holds a triage meeting at the beginning of each week to discuss open issues. Resolved issues are updated and the status is communicated to the client. Each resolved issue must have an API or SILK test that proves the issue can be closed.

Another aspect of our quality process is the daily build. At 4am every day, the application is extracted from the source control system, Perforce Software's [13] Perforce [14]. The system is built from scratch and automatically installed on a clean testing machine. All API and end-to-end tests are executed automatically. This process e-mails a detailed report and a summary of the testing process to the QE automatically. This procedure ensures that the software is always ready for release.

The full-time QE ensures that each release has acceptable quality. The QE manages issue tracking, system delivery, and end-to-end testing for the development team. The selection of the right project management tools aids the quality process. We have found that defining measurable quality metrics and granting release-blocking authority to the QE have improved overall quality and acceptance rates. An automated build and test system ensures that the software is always ready for release.

## REFERENCES


1. Beck, Kent, "Extreme Programming Explained," Addison-Wesley, 0201-61641-6.
2. Beck, Kent and Cunningham, Ward, "A Laboratory For Teaching Object-Oriented Thinking," presented at OOPSLA 89, available online at http://c2.com/doc/oopsla89/paper.html.
3. ClearQuest information available online at http://www.rational.com/products/clearquest/index.
4. Jacobson, Ivar, "Object-Oriented Software Engineering: A Use Case Driven Approach," Addison-Wesley, 0201-54435-0.
5. JUnit, open source testing framework, available online from http://www.junit.org.
7. NuMega, a subsidiary of Compuware Corp, online at http://www.numega.com.
8. Perforce information available online at http://www.perforce.com/perforce/loadprog.html.
9. Perforce Software Inc web site, available online at http://www.perforce.com.
10. Rational Software Corporation, online at http://www.rational.com.
11. Segue Software Inc web site. Online at http://www.segue.com.
12. SILK information available online at http://www.segue.com/html/s_solutions/silk/s_family.htm
13. Trilogy® web site. Online at http://www.trilogy.com.
14.TrueCoverage information available online at http://www.numega.com/products/cover/tc_java.shtm.