Essential Agile Engineering Practices
Participant Workbook

A Course from ThoughtWorks Studios

© ThoughtWorks 2011. These materials are for the sole use of the individual to whom they were delivered.

Introduction

Welcome to Essential Agile Engineering Practices, a course in the Agile Foundations series by ThoughtWorks Studios. This course is intended to help developers understand what it takes to be effective on an Agile project. There is coverage of the core Extreme Programming (XP) engineering practices, in addition to broader topics such as quality management, iterative and incremental design, and ongoing skill development.

By combining theory, hands-on practical exercises, and group discussion, participants will leave the course armed with a toolkit of principles and techniques to handle the practical matters of day-to-day Agile software development.

The audience for this course is primarily software developers, but can also include Quality Analysts, Software Testers, Enterprise Architects, and Technical Managers. All individuals should be prepared to participate in labs that involve programming. In addition, it is assumed that all students have taken the pre-requisite course, Agile Fundamentals by ThoughtWorks Studios, or have the sufficient background knowledge about Agile and it's fundamental principles and practices.

This course is two-to-three days long and consists of facilitated discussion and non-technical workshops, along with lab-based simulation of the Agile development lifecycle.

© ThoughtWorks 2011. These materials are for the sole use of the individual to whom they were delivered.

Learning Objectives

At the end of this course participants will be able to:

- Understand the core Agile engineering practices and how they impact the quality function on an Agile team
- Understand test-driven design / development by way of hands-on instruction.
- Understand refactoring by way of hands-on instruction.
- Understand continuous build and release practices by way of hands-on instruction.

What's in it for me?

In the space provided below, write down what it is that you personally would like to learn or achieve from attending this course.

© ThoughtWorks 2011. These materials are for the sole use of the individual to whom they were delivered.

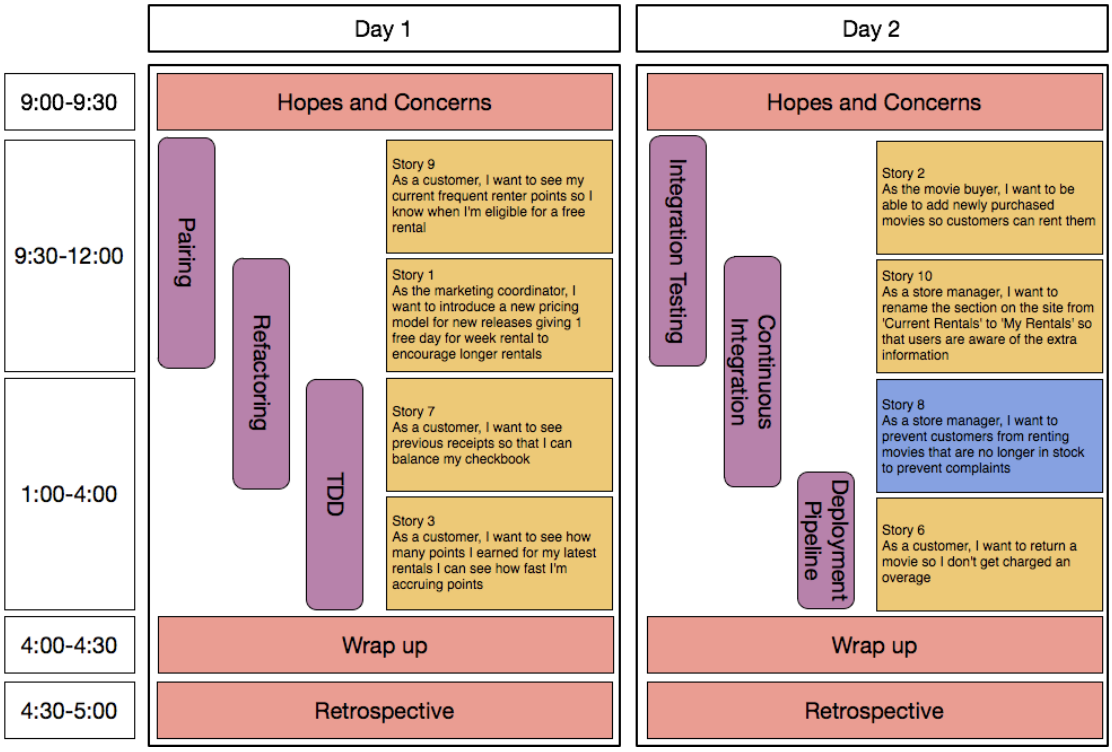
About the Learning Lab

The Agile Engineering Lab is focused on an existing application for an online video-rental business. Over the duration of the course, participants are presented with a series of user stories that detail new features to be added to the application. Agile engineering practices are introduced one by one as new challenges arise in the development effort.

Students will be working in development pairs and will be provided with a virtual training lab that contains all of the necessary development tools for the exercises. Typically, concepts will be introduced in a discussion context, followed by a demonstration by the instructor, and ultimately worked through by the students, both individually and as a collective group.

Many of the exercises are structured so that the students encounter a number of opportunities to exhibit anti-patterns. Sidebars will be conducted as these events occur on such topics as evolutionary design, self-documenting code, and the ability for developers to better determine story sizing based upon experience dealing with new requirements.

Lab structure



© ThoughtWorks 2011. These materials are for the sole use of the individual to whom they were delivered.

Course Topics

Tips for Refactoring

What is refactoring? Refactoring is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior. The intent is to improve the quality of the code. Refactoring should ideally be done as you are coding new functionality, working with legacy code or fixing bugs.

Steps

- Make sure you have good test coverage for functionality you are Refactoring
- Make sure all tests pass
- Refactor with a pair
- Run all tests until passing

Common Refactorings

- Rename variable, method, or class
- Extract variable
- Extract method
- Pull up or Push down
- Introduce parameter
- Introduce interface

Supporting Practices

- Collective code ownership
- Pair programming
- Test-driven development

Test-driven development

Uncle Bob Martin's Three Laws of TDD

1. You are not allowed to write any production code unless it is to make a failing unit test pass

© ThoughtWorks 2011. These materials are for the sole use of the individual to whom they were delivered.

2. You are not allowed to write any more of a unit test than is sufficient to fail; and compilation failures are failures
3. You are not allowed to write any more production code than is sufficient to pass the one failing unit test

Continuous-integration practices

- Check in regularly
- Create comprehensive automated test suite
- Keep the build and test process short
- Don't check in on a broken build
- Always run all commit tests locally after updating, before committing
- Never go home on a broken build
- Always be prepared to revert to previous revision
- Don't comment out failing tests/assertions
- Test-drive development
- Fail the build for slow tests
- Fail the build for warnings and code-style breaches

Continuous-delivery practices

- Only build your binaries *once*
- Deploy the same way to *every environment*
- Smoke test your deployments
- Keep your environments similar
- If anything fails, stop the line

© ThoughtWorks 2011. These materials are for the sole use of the individual to whom they were delivered.

Functional test suite practices

Few most important practices to keep in mind:

- Make sure that tests are independent
- Contexts
- Reusability

▪ UI Changes

- XPath is difficult to maintain – avoid it, if possible
- ID change: Does it make sense? Make sure to create constant and reuse
- Follow DRY (Don't Repeat Yourself) principle

▪ Flaky/Instable tests

- Page synchronization
- Explicit waits

▪ Other guidelines

- Write it from business point of view
- Reusability
- Refactor steps whenever necessary
- Extract concepts as you go
- Use contexts wisely – current limitations
- Cannot add/delete parameters
- Cannot change the order of contexts
- Thumb rule for workflow – not more than 10 steps
- Verification/Assertion not being part of concept/context

ThoughtWorks studios

© ThoughtWorks 2011. These materials are for the sole use of the individual to whom they were delivered.

Articles and References

Articles and Blogs

- Elisabeth Hendrickson's blog post on agile-friendly testing tools: http://testobsessed.com/2008/04/29/agile-friendly-test-automation-toolsframeworks/
- Page-Object Pattern by Simon Stewart: http://code.google.com/p/selenium/wiki/PageObjects
- Martin Fowler's Refactoring site: http://refactoring.com/
- Ward Cunningham's wiki: http://c2.com/cgi/wiki?PairProgramming
- Strangler pattern for legacy code: http://martinfowler.com/bliki/StranglerApplication.html

Support References

- Studios Community site: http://community.thoughtworks.com
- Video World code example: https://github.com/BestFriendChris/videoworld

Supplemental Reading

Test Driven Development

- Kent Beck, Test Driven Development: By Example
- Dave Astels, Test-Driven Development: A Practical Guide

Domain Language

- Eric Evans, Domain Driven Design
- Martin Fowler, Domain-Specific Languages

Pairing

- Ward Cunningham, Ward's wiki (http://c2.com/cgi/wiki?PairProgramming)

Refactoring and redesign

- Martin Fowler, Refactoring: Improving the Design of Existing Code
- Michael Feathers, Working Effectively with Legacy Code

Functional Testing

- Lisa Crispin and Janet Gregory, Agile Testing

Continuous Integration and Delivery

- Jez Humble and David Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation

Agile development

- James Shore and Shane Warden, The Art of Agile Development

© ThoughtWorks 2011. These materials are for the sole use of the individual to whom they were delivered.