**Agile Developer Practices**

**Facilitator Guide**

……………………………………………………………………………………………………

A Course from ThoughtWorks

………………………………………………………………….…………………………………

This work is licensed under the

Creative Commons Attribution-ShareAlike 4.0 International License.

To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/4.0/>.

Contents

Course Overview 3

Preparation 4

00. Opening Overview 9

Why Are We Here? (Hopes and Concerns) 10

01. Pairing 10

02. Pairing and Refactoring 11

03. Refactoring and Redesign 11

04. Test-driven development 12

05. Functional Testing 12

06. Functional Testing and Continuous Integration 13

07. Continuous Integration and Continuous Delivery 14

Retrospective / Plan for next day 15

Starfish Technique for Retrospective 16

Day Two 19

Wrap-up 20

Course Overview

## Learning Objectives

* To understand what it takes to be effective on an Agile project via the core Extreme Programming (XP) engineering practices, in addition to broader topics such as quality management, iterative and incremental design, and ongoing skill development

## Agenda

[Note: Refer to engineering-practices-agenda.docx.]



# Preparation

## Room, Facilities & Stationery

* **Projector**
* **Laptop** (optional)
* **Wireless presentation remote**
* **Desktops or laptops** with remote desktop capability and external keyboard and mouse
* **Good internet connection:** We use Skytap Cloud as virtual training setups. to access this setup. Hit the following url - <https://cloud.skytap.com/tools/connectivity> and click on "Start Test", which will tell you the status of your connection. At the end, this also collects some diagnostics, which you can view them by clicking "Diagnostics" in the small text box. I would encourage you to send us the diagnostics.
* **Flat-screen monitors:** Pairing is best experienced with large flat monitors with external keyboard and mouse (optional)
* **Wall space** to be able to post cards and flip-chart pages. If you don’t have any wall space, get an extra white board around 4 feet by 3 feet in size.
* **Tables.** Arrange class in the form of tables that can huddle 4 - 8 people on each table. Prefer to have a training room setup like one/more tables and attendees sitting around the table with enough space in the room. It helps the trainer to walk around the room to debug or fix code.
* **White board(s) or Flip chart Easels.** If you don’t have wall space where you can stick up flip charts, one board for each team (up to x people) works well. Can be small, around the size of a flip chart. This will be very handy when you are doing the Lego game and the teams manage their points and burn-up chart.
* **2 x 3M Flip chart pad, Unruled 25x30, White** (about 30-40 sheets)
* **Clips** to hold the flipcharts on the board if you are not able to stick them on the walls
* **Timer** (iPhone App Easy UP/Down Timers @ $3.99 is great for this)
* **Index Cards, 4x6, assorted rainbow colors**
* **Sticky Notes 3 x 3 Five Ultra Colors** in multiple colors. Handy for the retrospectives and capturing hopes and concerns.
* **Dry Erase Markers** in multiple colors. Green, Red, Blue, Black. Carry one set per team and one set for the facilitators.
* **White-board erasers** (sometimes this is missing and we end up using paper or tissues)
* **A set of sharpie pens**. Carry 1 per participant with a few extras.
* **3M Scotch Magic Transparent Tape or Blu-Tack** to stick up cards on the wall (something that won’t mark the walls, like blue painter’s tape)

### Things to print!

* **Feedback forms:** Make sure you have one for each participant and a few extras.

### General Guidelines

* As much as possible try to write notes and make points on flipcharts and not white boards. This makes it easy to capture the information and retain it for future reference as opposed to white boards where you have to erase it.
* When doing retrospectives or hopes and concerns, you may get more information if you ask people to write things on post-its as opposed to saying things out. Also, using post-its parallelize the thought process in many ways, so you may be able to process more information in the same time.
* Make sure you run through the deck, in “**presentation**” mode at least once before you do the session! See our guidelines for reusable presentations ([https://my.thoughtworks.com/groups/education/content?filterID=content~category[course-development-tool-kit](https://my.thoughtworks.com/groups/education/content?filterID=content~category%5bcourse-development-tool-kit)])

### Creating the Agenda Wall

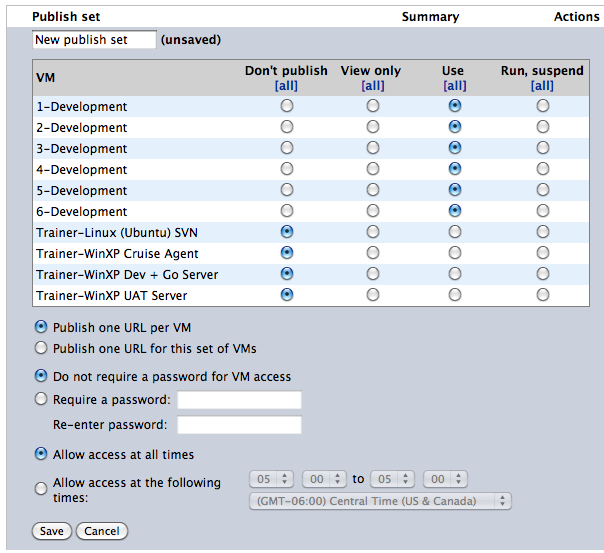
* We have found it very useful and interesting to use a card wall as an agenda. On white cards **write** down 4 statuses *(We say “write” as opposed to print, because doing so makes it more low tech and something that feels like it’s easier to change. It is closer to how we manage our backlogs in the physical world.)*
  + Backlog
  + In Progress
  + Done
* Use blue cards to write the headlines of all the different sections of the course. This can be picked from the Agenda directly.
* Use yellow cards to write out all the activities (exercises) that will be done in the course.
* Exact colors can change, but having a color coding gives a good feeling to the group on the mix of the course and helps us reinforce the simplicity and the power of the wall.
* Before the class begins, create the wall where it is visible to the whole group. Use this wall to actively move cards as we progress in the session.
* The In QA stage is just to ask the group if they have any questions on that section. If they don’t then move the card, else try to answer the question or put it on the parking lot if it is not within the context of the course.
* Markers in multiple colors can be used for:
  + Green to denote what went well in retrospective
  + Red for “Can do better” or “Areas for improvement”
  + Burn-up chart. Blue for planned scope, green for accomplished scope etc.

## Technical setup

### Prerequisites

* Access to the [EngPractices Training" project on SkyTap](https://cloud.skytap.com/)
* Updated Skytap configuration:
  + Latest version of Twist is installed
  + Eclipse has latest software updates
  + Twist has subversion

### Create the training environment:

1. Create a configuration on SkyTap, using the "Dev Practices (2012)" template (Library -> Select " Dev Practices (2012)" -> Create Configuration), and give it a name relating to the client, e.g. " Dev Practices (NCR 01-23-12)."
2. Optionally remove any unrequired sets of pair machines from the new configuration (if there are less than 8 pairs).
3. Share the configuration with the "Agile Developer Practices Training" project.
4. Publish a single URL for each pair (set of 4): choose Settings->Publish URLs and create a Publish set for each pair with all machines set to "Don't publish", except for the 4 for that pair which should be set to **Use**.  
   
5. Record the URL Details for each machine
6. Adjust "Auto Suspend" value to 2 hours:  choose Configuration Settings->Auto Suspend and set value to 2 hours, click Update button

### Start the Machines in the environment

1. Select "Return to Configuration" or "Configuration"
2. Ensure that all machines in the configuration are selected
3. Click **Run**.

### Test RDP connections & record RDP settings

1. After all of the machines have started, attempt to connect via RDP (login to each with admin/p@ssw0rd)
2. During connection attempts, record the host settings for each machine

### Initialize the SVN repository

1. Start the Trainer Linux Agent (& SVN server), and log in with the credentials trainer/p@ssw0rd
2. Execute the following:

tar zxvf /mnt/shared/twist/videoworld-setup.tgz  
  
cd ~/videoworld-setup  
  
./buildrepo  
  
sudo ./resetrepo  
  
sudo /etc/init.d/svnserve start  
  
./updatepairs svn://localhost videoworld/trunk 2

At this point you should be ready to begin.

## Student Machine Setup

Run the following setup script for each developer machine (recommendation: Set up one more developer machine than the number expected to need). The login for the student machines is admin/p@ssw0rd

### Downloading the initial code

1. Open Eclipse
2. Select File | Import | SVN | Checkout Projects from SVN, click Next
3. create new repository location, click Next
4. create connection to svn://svn-server, click Next
5. enter pair credentials(?) for SVN (and save password) [ credentials:  pairN/p@ssw0rd ]
6. import from svn repos (PairN | Trunk) for the pair station, click finish button
7. After svn checkout completes, close Eclipse

### Modify the Windows Environment

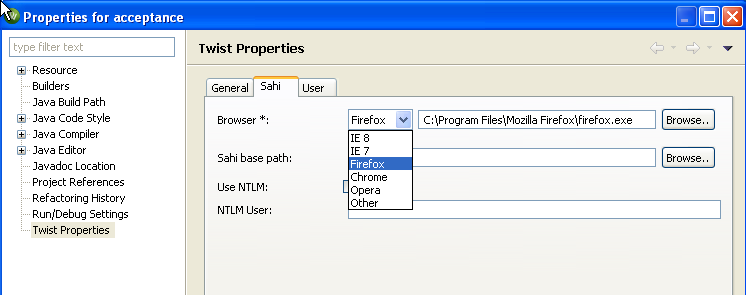
Modify Properties of cmd.exe (right-click cmd.exe) to start in c:\workspace\videoworld

### Launch & Configure Twist

1. Launch Twist
2. Select File > Import | General | existing projects into workspace, click Next
3. Select Root Directory of **c:\workspace\videoworld , click OK**
4. Click finish

      (need to perform another import to get the package structure right)

1. Select File Import | General | existing projects into workspace, click Next
2. Select Root Directory of **c:\workspace\videoworld\acceptance , click OK**
3. Click finish
4. In Twist, open Twist Properties and set the browser as Firefox (by default, it is IE8, which causes the bug of tests not closing the browser window when they finish running):



1. You may also need to migrate from an older version of Twist when prompted. Do so.
2. Close Twist

Recommendation: If you do the setup the day before the training -- After all machines have been set up, return to the Skytap configurations page and suspend all.

**Attendee requirements:**

1. Sample app is a typical business app – retail video rental. Some web development knowledge
2. Java web framework called Struts (knowledge of sitemesh, freemarker would be bonus)
3. Knowledge of Dependency Injection (knowledge of spring framework)

00. Opening Overview

## Slides summary

* Review agenda
* Facilitator introduction
* Why are we here?
* Learning objectives
* Agile/XP best practices (Organizational, Team, Individual)

### Starting the Class / Hopes and Concerns

* Start the class with a brief welcome to the group and then get into the introductions. Introduce yourself first. State the roles you have played in the past and how you have been actually contributing to teams as a practitioner on the ground. Allow the class to introduce themselves.
  + Name
  + What role(s) they have played in the past and what the current role is?
  + How many years in the industry?
  + What company/group/department they are from (in case you are in a public setup where you have multiple companies participating)?
  + What is their experience in/with Agile?
* Once the introductions are done, do the “Hopes and Concerns” exercise. See later for details on “Hopes and Concerns”.
* Introduce the agenda and format of the class. Walk through the agenda that you have already put up on the wall. As you walk through the agenda, tick off any items listed on the flipchart related to the hopes and concerns, that would be covered by those agenda items. If there are others on the list that are not in the Agenda, call them out as out of scope of the current class. Inform the class that, time permitting, you will go through the list at the end of the class and see if the hopes have been met, or the concerns relieved.
* At this time introduce the “Parking Lot”. It is a place where you will add all questions we may not be able to answer during the class, as they may be irrelevant to the current module, or outside the scope of the current discussion. Commit to the class that you will get back to this list and by the end of the class and either respond or follow-up after the course.
* Talk about the class being interactive and encourage the group to ask questions as we go. Also, make sure at the same time that you are deferring the questions to be answered in the right sections of the course. You will find that students ask questions on everything you are going to cover in the next two days as those questions come to mind. You do not need to answer them when they come up. You do need to record them in the Parking Lot.

# Why Are We Here? (Hopes and Concerns)

## Supplies / Preparation

* Blank flipchart pages for Hopes and Concerns
* Choose one of the two options by which this can be done. Option 1 leads the participants to reveal more hopes and concerns as they hear what others are saying. This uncovers more hopes and concerns as you proceed. It does require you to write them down, but this is effective in slowing things down and giving them a chance to think. Option 2 allows them time to think on their own, but it isolates them from each other in the process and does not encourage interaction until they start doing clustering on the board.
* Estimated time: 30 minutes

## Option 1: Without using post-its

* **The class calls out hopes** and concerns and the facilitator writes them on the flip chart as people call them out. Use Green marker for hopes and Red marker for concerns.
  + Once you have heard all the hopes and concerns, you could point to agenda and call out some of the hopes that will be met during the class. Others can go on the parking lot
  + For the concerns, as a group you can decide if there are any concerns that can be addressed by the facilitators during the class with respect to the sessions themselves. If there are larger concerns around the organization etc. lay out questions that can be answered to allay those concerns. Maybe add these questions to the parking lot.

## Option 2: Using post-its

* **Give the class 5 minutes to write up on post-its**. Hopes on green post-its and concerns on pink post-its. Get them to come over and stick up their hopes and concerns on the flipcharts in front of the class. While they are doing this, facilitate the group to place similar hopes and concerns as groups and summarize the list once done. Follow the same process of addressing these are in option 1.

01. Pairing

## Supplies and Preparation

* Update list of Skytap links in the slide near the end of the module.

## Process

* How to pair
* Pairing infrastructure
  + Monitors
  + Keyboard and mouse
* Pair-changing techniques
  + Pairing chart/”pairamid”
* Hands-on: Story #9 (display updated Skytap links on presentation slide)

02. Pairing and Refactoring

## Supplies and Preparation

* Skytap

## Slides Summary

* Pairing styles
  + Driver-navigator
  + Ping-pong
  + Ball and board
* Refactoring
  + What is it?
* Refactoring examples
  + Rename
  + Extract method
  + Split loop
* Hands-on: Story #1
* Hands-on: Story #11 (optional/if needed)

03. Refactoring and Redesign

## Supplies and Preparation

* Skytap

## Slides Summary

* Refactoring vs. Redesign
  + Point out the importance of keeping refactoring and redesign separate
  + Large redesigns are NOT refactoring
  + Be clear with the business owners that any redesigns are an investment in future speed and maintainability.
* Redesign tips
* Redesigning legacy code
  + Strangler pattern
  + Useful resources
* Hands-on: Story #10

04. Test-driven development

## Supplies and Preparation

* Skytap

## Slides Summary

* Why TDD?
* Uncle Bob Martin’s Three Laws of TDD
* Hands-on: Story #7

05. Functional Testing

## Supplies and Preparation

* Skytap, Twist configured

## Slides Summary

* Definitions: What is functional testing?
* Types of functional testing
* Example of an automated functional test
* Automating functional tests
  + Answers important strategic questions:
  + How can tests be executed early and often enough to mitigate risk?
  + How can tests be maintained as long-lived, evolving, and reusable assets?
  + How can testing involve all the stakeholders in the software development process such that requirements are fully understood and tested?
  + How can the highest value most essential testing be identified?
  + Functional test automation is not an end unto itself – it’s a tool and a capability
  + Automation does a few things very well:
  + Makes regressions run faster, freeing testers up to think, not click
  + Allows for running tests in different environments and on different platforms
  + Shores up the safety net
  + Extends your capacity to cover different testing models
  + Gives fast feedback to teams
  + Reduces the defects that make it to production by providing frequent regression checks
  + Makes requirements clearer by giving developers executable acceptance tests
  + Automation does not:
  + Find all of the bugs a user might encounter
  + Find logical inconsistencies, performance, security or UX problems in the application
  + Cover all boundary conditions, particularly at higher levels
  + Many organizations believe that unit tests are a panacea for software quality
  + Unit tests are good at a few things:
  + Preserving developer intention
  + Helping developers design code better
  + Helping developers think through boundary conditions
  + Unit tests aren’t good at:
  + Checking integration between components
  + Validating that software works from a user perspective
  + Finding unanticipated errors
  + Covering all boundary conditions
  + Brittleness
  + Maintainability
  + Time Required
  + Not inherently collaborative
* Strategy
  + Testing pyramid
* Hands-on: Story #2

06. Functional Testing and Continuous Integration

## Supplies and Preparation

* Skytap, Twist

## Slides Summary

* Test automation patterns
  + Page object pattern
  + Domain object pattern
* Functional testing best practices
  + UI changes
  + Unstable tests
  + Scenarios
* Continuous integration
  + Why?
  + The Build: Backbone of Agile Development Process
  + Benefits of continuous integration
  + Prerequisites
  + Practices
    - Always be prepared: What if someone DOES go home on a broken build?
    - Recommend Humble and Farley’s Continuous Delivery book
  + Suggested practices
* Hands-on: Story #8

07. Continuous Integration and Continuous Delivery

## Supplies and Preparation

* Skytap

## Slides Summary

* Traditional approach to delivery
  + How long would it take your organization to deploy a change that involves just one single line of code. – Mary and Tom Poppendieck
  + Do you do this on a repeatable, reliable basis?
* The ideal approach to delivery
  + Software always production-ready
  + Reliable deployments
  + Everyone can self-service deployments
  + Releases happen according to business needs
* Principles
* Deployment pipeline
* Stages and environments
* Practices
* Continuous deployment
  + Flickr example

Retrospective / Plan for next day

* Introduce and describe what a retrospective is. We do this to improve, whether it’s this course or the development team’s processes. Remind the group that they can use the following technique for their own retrospectives.
* Depending on how many questions get asked and how the day progresses, the amount of time we have for retrospectives may differ. If we have the whole 40 mins, then would be a good idea to do a “Starfish” retrospective. See later for a detail on how to run the Starfish retrospective.
* If we don’t have enough time, then a simple “what went well” and “what can be done better” using sticky notes would be good. Be sure to have action items from this, and if there are things that you could do to change the class for the next day, make sure you make that happen. That will truly showcase the power of the retrospective.
* Once done with the retro, if you are using the agenda wall, it will clearly show you what you have gotten done with and a quick review of the backlog of cards will be a good way to wrap up the first day of the class.

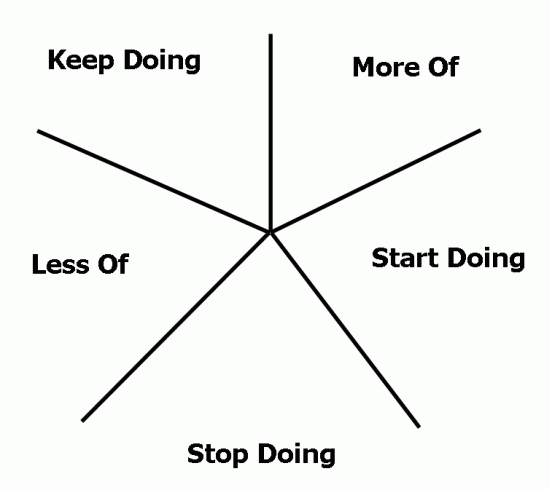
# Starfish Technique for Retrospective

## Supplies / Preparation

* A board/wall where we could draw up the starfish with enough space for teams to add sticky notes.
* Different colored sticky notes to represent each category. Its useful to find different colored sticky notes because when people put up their thoughts, many are related and they may actually put them in different sections of the starfish. When you try to group them in themes, its useful to know which part of the fish they came from e.g. stop doing, do more of etc.

## Process

* Draw up the start fish on the wall/board like in the image. Put up a different colored sticky note in every section so the group knows what colors to use for what section. (if you don’t have enough color sticky notes, you can ignore this part)
* Distribute the colored sticky notes and instruct the group to stick to the color schemes when they come up with points. Explain to the group a little of what each of those sections mean. See later for the sections.
* Once they are aware, give them 10 mins to write stuff up on sticky notes and get them to stick it on the wall as and when they are done. Help them stick things on the wall if they are too busy writing.
* While sticking the notes, try to group similar notes within the same category. Sometimes these could be across categories and thatz where the color coding comes in handy.
* Read out some themes to the group based on what is collated. Calling out some interesting and relevant inputs helps.
* If you have time, prioritize one or two of the main themes and have a quick 3 mins discussion on each of those.
* If possible, come up with action items. Try to limit to a handful, but if you have many, use dot voting to prioritize (keep expectations realistic for day 2).



* **Keep Doing** - Is a good starting point for team members to focus on typically all the good things that they liked about a project. You might want to encourage people to think about things in terms of, what would they miss if they didn't have a particular practice, technique, technology, person, role, etc. A good example from a real session I've been in before is 'Running performance benchmarking and tuning during an iteration helps to identify regressions or slowdowns so we can address them earlier'.
* **Less Of** - Helps to focus on practices that might need a bit more refining or that were simply not helpful in the current circumstance. Perhaps they add value but not as much as other practices could. An example here is that perhaps stand ups have become status meetings and so there should less of talking to one person (and more of talking to each other) during them.
* **More Of** - Is another type of focus that helps further refine or highlight practices, technologies, etc that team members might want to try more and are not necessarily taking full advantage of. A good example is that maybe people are pair programming but knowledge transfer and a better understanding of the code changing might be gained by doing more of swapping programming partners.
* **Stop Doing** - Obviously for things that are not very helpful to development practices or not adding much value. Perhaps it's about writing that status reporting email at the end of the day (because you can substitute a simple one minute conversation for it instead)
* **Start Doing** - Is a great opportunity for team members to suggest new things to try because of things that may not have gone so well or just for simply keeping things dynamic and fun. Perhaps you might want to try a burn up chart on the whiteboard or try some new open source tool for helping improve developer productivity.

## Objective

* Diagrams are always useful focal points for starting discussions, and that's one reason I like using the starfish diagram for a retrospective. This particular retrospective technique helps people by getting them to reflect on varying degrees of things that they want to bring up, without having it fit into the black or white category of 'What Went Well' or 'Not So Well' so I think it scales a little bit better.
* Getting people to either write things up under the starfish in this manner gives you a scattergram of sorts and is a great visual technique of estimating the overall health of your project. Most of the points on the starfish also try to coerce people into actually creating action items instead of simply saying that something was not good.
* Note: The time frame for the retro is too short to do a full-fledged start fish retrospective. So, the idea would be to gather information as inputs, read out a few and get them introduced to the technique of doing starfish retrospective.

# Day Two

* Start the day two with a quick word recall. It’s a way to refresh the group’s mind on what they learnt yesterday and getting them back into the context of the class. Also, this highlights how many new things they have learned in the one day. It generates good excitement in the group. If you see that not too many are coming up, ask them leading questions so they start giving the answers. E.g. “what units did we estimate in yesterday”, “what did we use to track progress of the project” etc.
* Once done, this can lead into the standup exercise. See the standup exercise for more details.
* Review the agenda and learning objectives for the day (?).
* Review the action items and decisions from the retrospective (?).

Wrap-up

## Visit the Parking Lot

* At this point, many a times we find that we have over run on time. If we have time, would be good exercise to run through the parking lot and try and answer the questions. Most of these turn up to be discussions in the group. If we have more time, we can encourage the group to come up with more questions and try and answer them was we go.
* If we cannot answer any of those questions, I like to carry it back and shoot and email with some pointers to the group.

## Retrospective / Feedback Forms

* At the end of the final day, we need to collect the feedback form the group. So make sure that we get the feedback forms out and get the group fill in the feedback. If time permits, we can do a quick what went well and what could be done better retro for day 2.
* Leave the audience with a small set of concrete action items (one to three) and a plan for how to implement each.

## Close

* Well, say thank you and good bye ☺. Leave business cards and/or contact information.