Coding Dojo: an environment for learning and sharing Agile practices

Danilo Sato
ThoughtWorks Limited
dsato@thoughtworks.com

Hugo Corbucci, Mariana Bravo
Department of Computer Science
University of São Paulo, Brazil
{corbucci, marivb}@ime.usp.br

Abstract

Resumo...

1 Introduction

> In software we do our practicing on the job, and that's why we make mistakes on the job. We need to find ways of splitting the practice from the profession. We need practice sessions.
>
> –Dave Thomas

The idea of a *Code Kata* was initially proposed by Dave Thomas as an exercise where programmers could write throwaway code to practice their craft outside of a working environment [6]. Laurent Bossavit later proposed the idea of a *Coding Dojo*: a session where a group of programmers would gather to solve the *Code Kata* togheter [4]. Although the session is organized around a programming challenge, the main goal of a *Coding Dojo* is to learn from others and improve design and coding skills through deliberate practice. This creates a learning environment where Agile technical practices, such as those proposed by Extreme Programming (XP) [3], can be shared.

This report describes the authors' experience of founding and running a *Coding Dojo* in São Paulo, Brazil. Section 2 will present the concept and rules of a *Coding Dojo* and the tailored process to conduct the sessions in São Paulo, improved over time by retrospectives. Section 3 will present lessons learned from the weekly meetings being held since the first session in July, 2007. Section 4 will discuss the aspects of a *Coding Dojo* that foster learning and tacit knowledge sharing, concluding in Section 5.

2 Coding Dojo

A *Coding Dojo* is a periodic meeting (usually weekly) organized around a programming challenge where people are encouraged to participate and share their coding skills with the audience while solving the problem. The main principles of the *Coding Dojo* are to create a **Safe Environment** which is collaborative, inclusive, and non-competitive where people can be **Continuously Learning**. Some of the XP principles align nicely with that [3], such as **Failure** – it is OK to fail when learning something new – **Redundancy** – one can always gain new insights when tackling the same problem with different strategies – and **Baby Steps** – each step towards the solution should be small enough so that everybody can comprehend and replicate it later.

There are some general rules that allow the *Coding Dojo* session to be productive and to flow. The meeting is held in a room with enough space for all the participants and usually requires only a projector and a computer or laptop. Having whiteboard space for sketching and design discussions is also valuable. The participants are encouraged to develop the solution using Test-Driven Development (TDD) [2] and are free to choose whichever programming language they prefer. There are two main meeting formats:

- **Prepared Kata**: In this format, someone has already solved the proposed *Kata* prior to the meeting (alone or in group) and the solution is presented to the audience during the session. Instead of showing the final code and tests, the presenters start from scratch, explaining each step and allowing the other participants to ask questions or make suggestions. The session goal is that everyone should be able to reproduce the steps and solve the same problem after the meeting.
- **Randori**: In this format, the participants solve the problem together, following TDD and Pair Programming in time-boxed rounds (usually between 5 and 7 minutes). At the end of each turn, the pilot joins the audience, the co-pilot becomes pilot, and a new co-pilot joins the pair from the audience. An extra rule is that discussions and suggestions should be only given when the pair arrives in a green bar, with all the current tests passing. The reason is that, while on a red

bar, the pair should focus and work together to get the tests passing. The audience can always suggest refactorings and optimizations on a green bar.

These formats allow the creation of an environment where participants can discuss and practice a wide range of topics, such as: TDD, Behaviour-Driven Development, Agile, Refactoring, Pair Programming, Object-Oriented Design, Algorithms, different programming languages, paradigms, and frameworks.

2.1 Coding Dojo@SP: Numbers and Processes

session agenda, # of participants, # of meetings, Languages utilized, ...[5]

3 Lessons Learned

Starting is always a problem. It is hard to choose what should be done first and what should be done later. Problems even raise when choosing what should not be done at all. Since the sessions are being held, the authors could identify what practices and rules went well (Subsection 3.0.1) but also found out that some things work less well (3.0.2) than expected. Finally, applying the practice to different audiences and in different contexts, the authors discovered some unaddressed issues (3.0.3).

3.0.1 What Went Well?

The goal is not to finish Information radiators Communication Inspiration for the meeting

3.0.2 What Went Less Well?

Moderating brazilians (hard not to speak on red) TDD/BDD and algorithms Balancing randoris and prepared katas Programming environment

3.0.3 What Puzzles Us?

How to reach a wider audience? How to share our efforts with the community? How to keep attendees engaged?

4 Dojo and Learning

Dreifus Model Deliberate Practice http://graphics8.nytimes.com/images/blogs/freakonomics/pdf/DeliberatePractice(PsychologicalReview).pdf Collaboration and Self-Organization Creating and sharing knowledge No single Master

One of the attendees joined the sessions just at the beginning when he had just finished his first semester in Computer Science. He is now on his third semester and most of his works are done using TDD no matter what language is being used. His latest work involved implementing sparse matrixes with common operations in C. He decided alone to do it using TDD with a testing library implemented during a *Coding Dojo* session ([1]). It resulted into an amazingly clear code with full test coverage. His ability to identify and pin down the required tests to force the correct implementation far surpass his collegues'. He has shown us strong evidences that the knowledge and practices adopted during the *Coding Dojo* can be absorved and understood regardless of experience on the matter. It also shows that the informal, non-directed and non-rigid format proposed can be very effective and supportive to more traditional teaching methods.

5 Conclusion

References

[1] C. D. S. P. S. 31. A short c unit testing library. `http://dojo_sp.googlegroups.com/web/31-CTEST.zip`, 2008.

[2] K. Beck. *Test Driven Development: By Example*. Addison-Wesley Professional, 2003.

[3] K. Beck and C. Andres. *Extreme Programming Explained : Embrace Change*. Addison-Wesley Professional, 2nd edition, 2004.

[4] L. Bossavit. Object dojo. `www.bossavit.com/pivot/pivot/entry.php?id=207`, 2003.

[5] Coding dojo wiki. `www.codingdojo.org`, 2007.

[6] D. Thomas. Code kata: How to become a better developer. `codekata.pragprog.com`, 2003.