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 [3]. 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) [2], 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 [2], 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) [1] 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: History and Process

The meetings in the São Paulo *Coding Dojo* started in 12^{th} of July, 2007 and have been held weekly since then in the Institute of Mathematics and Statistics of the University of São Paulo. Some extra sessions were done during the University holidays and most of the session reports are available on the international *Coding Dojo* wiki[5]. The number of participants varied from 3 to 16 and their skill level ranged from undergraduate students to experienced programmers.

On the first meeting the participants were asked to fill index cards with their expectations and personal interests in attending the sessions. An affinity map was built with that information and the three main interests were to practice problem solving skills, to learn different ways and algorithms to solve the challenges, and to learn new programming languages. Some of the sessions were highly focused on design problems and algorithms, which left less time for writing code, but the participants liked to learn from the discussions. On the other and, the majority of the sessions required less design and algorithms discussion, leaving more time to write code and allowing the participants to experiment with a wide range of programming languages, such as Java, C, Ruby, Python, Lua, and Smalltalk.

The sessions usually follow the same process:

- **Problem Choosing**: Before the meeting, the participants receive an e-mail with 3 to 5 options of problems to be solved. The problems are chosen from several sources (such as Ruby Quiz[1], Programming Challenges[2], UVa[3], and SPOJ[4]). Each option is briefly presented and the participants vote on which problem will be solved in the meeting. This usually takes 5 to 10 minutes.

- **Problem Discussion**: Once the problem is chosen, the group discusses the different approaches to solving it, usually ending up with an agreed approach and a list of TO-DO items, as proposed by Kent Beck [1], to guide the pairs during the implementation. This usually takes 10 to 20 minutes, but there were meetings when the group spent the entire meeting discussing algorithms and possible approaches to a complex problem.

- **Coding Session**: With an agreed approach to solve the problem, the participants start the coding session in one of the two formats – a *Prepared Kata* or a *Randori*. They should practice Pair Programming and Test-Driven Development as a general rule. This usually takes 1 to 2 hours.

- **Retrospective**: At the final 10 to 20 minutes of the session, the participants stops coding (even if the problem was not completely solved) to reflect on the experience and share the learned lessons with the group. This is also a good time to discuss what could be improved and to come up with action items for the next meeting.

Finally, the São Paulo *Coding Dojo* came up with two special roles that can be rotated between participants, but that are very important to organize and to make sure the meetings continues to happen. The **Moderator** or **Organizer** is responsible for what happens before, during, and after the meeting. He handles tasks such as reserving the meeting room, sending reminders and options of problems to be solved, setting up the computer and projector prior to the meeting, moderating discussions, conducting the retrospective, and cleaning up the room after the session. The **Scribe** is responsible for publishing the results of the session and sharing it with the people that could not attend the meeting. He handles tasks such as posting the session report to the wiki, publishing the final source code to the group, sometimes taking photos, and documenting the results of the retrospective.

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

[1] http://www.rubyquiz.com/
[2] http://www.programming-challenges.com/
[3] http://acm.uva.es/p/
[4] http://www.spoj.pl/

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

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

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

[4] Coding Dojo São Paulo - Session 31: A short C Unit testing library. `dojo_sp.googlegroups.com/web/31-CTEST.zip`, 2008.

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

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