

AntiPractices: AntiPatterns for XP Practices

Yoshihito Kuranuki, Kenji Hiranabe
TIS, Inc., Eiwa System Management, Inc.
kuranuki@sunny.tis.co.jp, hiranabe@esm.co.jp

1. Abstract

When you introduce Extreme Programming (XP) to your project, the team gets fresh energy and high efficiency. However, sustaining the good condition becomes difficult when you are attacked by its side effects. We call these XP side effects "AntiPractices", just as AntiPatterns come from Patterns. "Turning all the knobs up to 10" is the XP way, but AntiPractices show bad symptoms of the overdrive. (There is a proverb "more than enough is too much" in Japan.)

In this experience report, we identify AntiPractices discovered from our projects. We add prescriptions so that XPeres can share the countermeasures.

2. Introduction

In short, XP AntiPractices are "Frequently Made Mistakes in XP Projects."

Over the past years, I (Kuranuki) have introduced XP into several software development projects in earnest. Although one of them was our first XP trial for a mission critical enterprise application, the project was very successful. I believe it is because the members of the team were full of motivation, I as a coach studied XP enough and prepared for it, and the size of the team was also appropriately small.

In the project, I adopted as many practices written in the XP book [Beck99] as possible, but at the same time, I gradually customized the practices after each iteration so that they work with our project and with Japanese business and cultural style. Along the way, some members occasionally reported difficulties and sometimes the atmosphere of the team became bad. As I performed certain actions and solved the problems, I discovered that difficulties happen when you naively turn the XP knobs too high (there is a Japanese proverb "more than enough is too much."). It is like some side effects of strong medicine. We need to know how to behave when you are attacked by the side effects.

In this experience report, I present problems I have survived in our XP project, with the context and the countermeasures in a narrative story format. Then in the reflection section, I generalize them in a pattern-like format. We call them "AntiPractices", hoping that they can be shared in the XP community.

3. Project Profile

This took place in a contract software development, which deals with commodities futures. The server of the system was in the user company's headquarters and there were eight clients placed in branches. The system was mission-critical (24 hours 365 days). The contract was divided in two phases, the first for five months and the second for four months. We ported and extended a legacy system with web-service interface and rich client.

The team consisted of 7-9 members and worked together in our company offices.

Table 1 Project team members

Name	Experience yrs of (SD ¹ /OO/XP)	Role
KO	Many/1/0	Manager/Analyst
YK	4 /8/2	Coach
RH	3/2/0.7	Programmer/Architect
SN	3/0.2/0.2	Programmer/DB Admin.
JS	1/0/0	Programmer
MN	1/0/0	Programmer
IN	3/2.5/0	Programmer
HK	3/1/0	Programmer
TK	2/0.5/0	Programmer

4. Organization of AntiPractices

In the following sections, we present three AntiPractices we identified in the project described above (I should add that each AntiPractice has been verified by other projects in Japan). Each AntiPractice is illustrated in a near-verbatim story in four 'acts', named [Story], [Action], [After], and [Crystallize]. In the [Crystallize] section, we reflect and generalize the lessons learned into an AntiPatterns-like form [Brown98].

Table 2 Form of AntiPractice

Name	The name of the AntiPractice
Background	The context where the symptom occurs
Symptom	Visible bad phenomena
Cause	Why the symptom occurs
Ideal	What it should really be
Prescription	Countermeasures that stop the symptom by attacking the cause and making it close to the ideal, including prevention.

¹ Software Development

5. Brownie's Works – “The boss refactored our code!”

[Story]

MN a freshman, began to learn the fun of programming. JS also a freshman, had quite a good experience of programming in the university.

In the stand-up meeting one morning, they decided to try the first "freshman pair." They started programming, with fun. The coach approached and asked "Is everything OK?" they said "OK!" The coach knew that letting them alone was an important thing for their start.

Late in the evening, they checked in their code to their code base. The "fanfare" sounded nicely (their auto-build/test batch played fanfare if it is successful). "It worked! It was a hard one, thank you for your help, thank you... let's go home." They wrapped up their work and went home with satisfaction.

Late at the night, the senior programmer IN was working alone. "What's this messy code!!!? This supposed to be like this..." He refactored all the code the freshman pair had just checked in.

The next morning, MN and JS came to work and found their code gone! "Our code is gone!" IN said "Yes, I refactored the code. That's Collective Ownership!"

MN and JS were really depressed.... **"The boss refactored our code!"**

[Action]

The coach noticed their change. He caught IN (the senior) and talked with him alone. "I think you should have pair programmed when you refactored their code." "What if they are not there?" "Mail to the list why and how you refactored." IN is a craftsman. He didn't think it was really needed. But, he thought, this kind of thing would happen repeatedly, so it would be easier to teach them once. He started pair programming with juniors of his own accord and showed how to refactor the code.

[After]

IN's action made a new move in the team dynamics. MN and JS learned a lot from IN with pair programming. The team got back its normal or even better condition.

[Crystallize]

We named this experience "Brownies Work"² and prevented its repetition. A direct cause of this problem was IN's violation of pair programming. Although this can be seen as an overdrive of Collective Ownership, the root cause was that IN lacked a sense of team building.

Table 3 Brownie's work

Name	Brownie's work (The boss refactored our code!)
Background	Junior members actively write and commit code. Collective Ownership is working.

² The name is after a fairy tale. Brownies come at night, and they program for you by the morning while you are asleep.

Symptom	Juniors' proud source code is changed completely different by a senior at night. The new code might be better but their motivation goes away.
Cause	A side effect of Collective Ownership and violation of pair programming by Refactoring alone. The root cause is the senior's lack of sense of team building.
Ideal	Juniors get energetic to see their code working in the customer's system.
Prescription	Seniors should refactor the juniors' code in a pair programming session. If the senior has no choice, he should tell the team the reason and the process of his decision so as to make the team learn from it.

6. Anybody Syndrome - “I'm not necessary here”

[Story]

Two months after we introduced XP, everything seemed to be working great. Pair programming and Collective Ownership were working and the honeymoon number³ was getting high. Even if somebody had got a cold and took a few days off, the project would have gone on.

One morning, NK got a bad cold. He called in to the office and said to the coach "I think I have a cold. May I take a day off?" The coach said "Sure, it was a tough iteration. Don't worry about the team. Take a rest."

But the next day, he was not well enough, and the day after that. After four days' off, he came to the office and saw the team working just as well as the day he left. He was happy that everything was fine, but felt some loneliness. Something was uncomfortable. "What is my value for the team?" After that, he began to stay away from his work quite often. He murmured **"I'm not necessary here..."**

[Action]

The coach had an interview with HK face to face. "You are frequently absent from the office these days, is anything the matter with you?" HK answered, "when I took some days off and came back, I found that the team did not have any trouble at all." "Yes, that's a good thing!" "That means the team is fine without me." "That's not quite right. Everybody wants to work with you, and you have your own goal in the project. You participate in the project with that goal, don't you?"

The coach and HK talked about his goal in the project and his career plan. HK was grateful to the coach for giving an opportunity to talk over these kinds of things in a big picture.

³ Japanese XP community uses a pet coined term *honeymoon number* for Jim Coplien's *truck number*.

The coach started to have such conversations with one member after another on a face-to-face basis without a formal interview session format. He also started going home with members and having a drink on the way home from time to time.

[After]

The project members found that they had their own goals as well as the project goal and they were free to talk about them anytime. That became a grand rule set by the coach. They now talked about their dreams, plans, and families. They started acting more autonomously, and the fresh energy came back to the team.

[Crystallize]

We called this situation “Anybody Syndrome.” This can be seen as an overdrive of knowledge and role sharing by Collective Ownership and Pair Programming. But we found that once this started to occur, lack of one-to-one communication and self-concerns problem were there.

Table 4 Anybody syndrome

Name	Anybody syndrome (I’m not necessary here)
Background	The team shares information at a high level. Project goes well even if someone is missing.
Symptom	Some members suspect they are not needed in the team and they lose objectives and motivation.
Cause	Insufficient communication between the coach and members. The root cause is too much focus on information sharing and negligence of individual’s goals aside from the project.
Ideal	Information is well-shared and at the same time the members feel their necessity for the team.
Prescription	Even if the development is a team work, the coach shouldn’t forget to talk personally with individuals. Do interviews and objective management to the members periodically and catch the sign early.

7. Pairing Prison – “I’m always under observation!”

[Story]

RH was a senior programmer with three years of experience in waterfall type development environment. He had been interested in XP, and this was his first XP experience in a real project. He had wanted to try pair programming, so this was a wonderful opportunity for him.

For the first several weeks, pair programming was fun. Although he felt tired after work each day, his sense of satisfaction was much bigger than the tiredness. But pair programming became burdensome little by little. When he paired with juniors, he got serious so he become a

good example. With seniors, he felt like he was taking an examination. Pair programming made him feel observed in a prison. He could not stand the situation anymore, and consulted the coach. He said, **“I’m always under observation!”**

[Action]

The coach listened to RH. He found that they had no time to work alone and that it was not easy to take breaks while in pairs. The next day, the coach told the team that he recommended to take breaks more often (for industrious Japanese, it needs courage both to declare and to accept this). The team adopted this as a ground rule. In addition, they doubled the lunch time for their personal time.

[After]

RH got more enterprising than before. The team got energetic, again. Some pairs spent break-time together to increase their communication, some enjoyed their private time, and others (senior-junior pair) sometimes had private lessons. Longer breaks worked.

[Crystallize]

Pair programming needs an appropriate amount of breaks. We called this experience the “Pairing Prison” and memorized the situation.

Table 5 Pairing prison

Name	Pairing prison (I’m Always under observation!)
Background	Always pair program in development time.
Symptom	Some members feel observed anytime and uncomfortable. Then hate to come to the office every morning.
Cause	The down side of the pair programming (full time review). The root cause is the lack of respect for person’s freedom.
Ideal	In order to increase efficiency and mental health of pair programming, private mail check time and frequent breaks should be highly respected.
Prescription	Spare sufficient time for breaks between pair programming sessions. Balance breaks and work time. For example, double the lunch time and declare the time as private time. Take breaks frequently. ⁴

8. Reflection

To understand and explain the idea of AntiPractices, we chose “disease” as its metaphor. For example, when you get influenza, first you find the symptom of cough or fever. The background may be that you have had a series of overwork, but the root cause is the influenza virus. So

⁴Kent Beck recommends drinking water frequently when pair program so that you bring up a break to go naturally. I would name this “Nature Callback.”

in order to attack the root cause (virus), you need to prescribe the appropriate vaccine to get the ideal state (healthy life) back.

Table 6 Disease and AntiPractice

AntiPractice	Disease
Name	Influenza
Background	Series of overwork
Symptom	Cough, fever
Cause	Virus
Ideal	Healthy life
Prescription	Vaccine

Just like disease, in order to find effective prescriptions, it is essential to find the root cause. You cannot stop the symptom just by trying to eliminate it. You have to analyze the symptom from the background, and to identify the cause. Then, you prescribe. This prescription often becomes another new practice to the team.

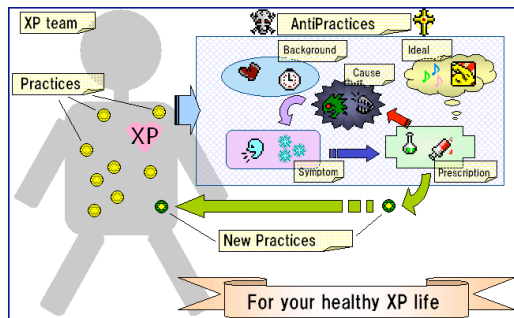


Figure 1 Prescription as a new practice

In the introduction of XP, we believe that you should let your own practices emerge according to your context over time, and that it is the heart of agility.

Besides the three AntiPatterns described in this paper, the following is our growing list⁵ of AntiPractices;

- Powerful Owner - "We are under his thumb!"(The person who's writing on the whiteboard has a power)
- Outnumbers' View - "We are a minority group"(Opinions of majority beats minority)
- Imperial XP - "Why do I do such a thing?"(Member believe that XP is always the only right process)
- Loose Control - "Sure. XP is free, you know."(Do XP-like practices without management, at all)
- Frozen Pair - "The two are pairing ... again!"(The same two members always pair until they finish their task)
- Formal Format - "Do we have to do all the practices just as in the book?"(Members adopt and fix the practices and stop improvement)
- Misleading Influenza - "I heard you are doing THE XP"(The effect of XP is promoted in a organization without its objectives, values and principles)

⁵ <http://www.ask.sakura.ne.jp/object-garden/>

The problems we encountered when introducing XP were not about technology. We found that the problems look like overdrives of practices but their root cause is about the people, the team, and the organization. Those problems are not easy to see until someone finds them and names them. AntiPatterns capture the problems by naming them like "Patterns". The problems caused by the practices are not solved only by adjusting and adding practices. The coach has to turn his eyes to the subtleties of people.

I first hit upon the idea of AntiPractice in the project, and started to collect them and to name them. After the project, I spoke about the AntiPractices in a workshop of the ObjectClub Christmas Conference⁶ held in December, 2003 in Tokyo.. In the workshop, we discussed the idea with other XP practitioners in Japan, and refined them.

As a final note, we'd like to describe the current situation of agile software development in Japan. The agile movement in Japan is not in the main stream yet and not many experience reports have been published. There are approximately 700,000 software development engineers in Japan and the XP white book [Beck99] has sold 10,000 copies. The XPJUG has 1,741 members, The Agile Process Association consists of 46 company members and the XP-jp mailing list has 3,000 members. We guesstimate XP and other agile processes are used in less than a few percent of the entire software development projects.

We have just started to collect AntiPractices in the Japanese XP community⁷. We are waiting for your contribution.

9. Acknowledgement

We wish to thank Alistair Cockburn for his advice on this narrative format ([Story] – [Action] – [After] – [Crystallize]) of this paper, which made each AntiPractice convincing and easy to read. We also thank our XP team members for their cooperation and hard working on the project.

10. References

[Beck99] Kent Beck, *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 1999

[Brown98] William Brown, Raphael Malveau, Hays McCormic III, Thomas Mowbray, *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*, John Wiley & Sons, Inc, 1999

⁶ <http://www.ObjectClub.jp/>

⁷ <http://www.ask.sakura.ne.jp/object-garden/>