Surviving in a Hostile Environment

Hans Wegener

Credit Suisse Financial Services, Postfach 100, 8070 Zürich, Switzerland
Phone: +41 (1) 334 66 51, Fax: +41 (1) 334 50 60, Mail: hans.wegener@csfs.com

Abstract: The IT departments of large enterprises usually enforce more rigid and formal development processes than Internet startups. Adopting a lightweight development process can become very difficult in such an environment. The reasons are both structural and cultural, and we describe how they can be addressed.

1. Introduction

Not all of us are born lucky. Extreme Programming is a development process that sometimes meets with resistance and cultural incompatibilites that are difficult to surmount. Especially in large enterprises, where rigid and formal processes prevail, it can be very hard to practice XP. Stilly trying to adopt it becomes tedious, frustrating, exhausting, because after all, we all believe in XP, don't we? The only conclusion seems to be that it is time to quit. As Jim Highsmith put it at OOPSLA 2000: "If your company has problems recruiting [or retaining] fabulous people, then it's because your company sucks." The largest are usually also among the most risk-averse and seem to be the last likely to adopt XP.

1.1. My Way Or the Highway?

While it is easy to look down in pity at sad figures who do not quit and keep getting frustrated, some thought should be spent on the question: "Do we really have to be so strict about the issue? My way or the highway?" It is desirable to be practicing XP in its pure form. But during the transition to XP it is sometimes difficult (if not dangerous) to make strict and strong statements. For example, managers who are confronted in such a way usually claim that it is "impossible" to work like that on their particular project. There is a legion of reasons you can find to dismiss proposals for practicing XP. Fundamentalism won't help if you want to change the way the world is turning. How can on make a step in that direction while staying faithful to most of XP's ideas?

I work in a large Swiss bank. According to a cynical remark from some extreme forerunner, my work environment could be paraphrased as the XP antichrist. Our team designs and implements the metalevel architecture for the bank's data warehouse. We started off a year ago with the task to establish metadata management by means of an (object-oriented) service architecture, web-based tools and associated management processes. There were no clear specifications around, just ideas and visions. It was the epitome of an extreme project [1]:

- small team,
- unclear technological basis,
- unclear requirements,
- time pressure.

On the other hand, what we also found was a standard development process as known from the late 80's: big design up front, time and scope are fixed in advance, etc. The process for bringing software into production is geared towards host-based applications, which makes it relatively tedious for us. The situation was complicated by the fact that we did not have just one customer, but two handful of them. They were distributed all over the bank and could be found both in IT and banking. In a word, this can be considered an environment hostile to XP. My way or the highway?

1.2. Success Is Possible, Even In a Hostile Environment

After having successfully brought four releases into production, we can confirm it was an unnerving and at times frustrating experience. However, we also clearly succeeded, since we:

- completed on time (no deadline slipped),
- got rapid feedback (one release each quarter),
- delivered high quality (so far no bug reports).

All releases were perfectly compliant with the company’s regulations. We stayed faithful to the bank’s process.

Our situation might not be representative of all development projects in large enterprises, but we have made our transition towards XP even in the face of adversities. Therefore, we would like to share some of our experiences.

2. Forces Working Against XP

Know the enemy. Understand that even in a beaurocracy there is a reason for everything that is done—even if it is as stupid as you can imagine. Getting behind the scenes and understanding why things work the way they do is the first step to get around them. Here's some of them.

2.1. Schedules and Budgets

Schedules and budgets are ubiquitous in large enterprises. It is difficult (if not impossible) to measure success and profitability of IT projects in a transparent and meaningful manner. Budgets are used to restrict the damage you can do, while schedules are used to determine what you are obliged to do. The two work together to control the direction the organisation is taking.

The interesting observation about schedules in IT is that once they are defined, it is rare for people to control fulfillment as long as the customer is happy. One reason is culture: project retrospects are not a common practice. Another reason is complexity: how do you define success if not by means of customer satisfaction? Most projects with XP characteristics do not define metrics for measuring the degree of goal fulfillment. Interestingly, meeting budget requirements can and is indeed measured.

We further define schedule as scope delivered within a timeframe. The schedule says what is delivered when. Unfortunately this contradicts typical XP project characteristics: you set time or scope, but not both.

Again, time can be measured quite nicely, so it is not uncommon that the customer enters your office on deadline day asking: "Where's my new release?" However, it is less common that he or she enters the room asking: "Did you change the push button in dialog Foobar to go gray instead of blue?" If the customer asks that, it is usually because he or she really cares about that requirement.

2.2. Division of Labour

For large organisations to scale properly, division of labour is almost unavoidable. So far people have not come up with a different yet scalable way of organizing collaboration.

In large IT organisations we usually find (at least) development and production support departments. Development produces software (design, implementation, test) that is later operated by production support (surveillance, administration). To ensure the quality of the delivered software, a change control board controls changes to productive systems happen by means of a formal test procedure. Usually this procedure is tedious and involves (a lot of) red tape, mostly paper.

The observation here is that paper is taken as sort of a proof of quality. In the absence of an understanding of the software (which is a consequence of the division of labour), change control boards (have to) trust procedures more than they trust statements. This leads to an interesting phenomenon: as long as you follow the procedure, it is alright to "shoot yourself in the foot". It is rare that measurable metrics are applied (for example, test coverage).

3. Transitioning to XP in the Face of Adversities

XP is built on a value system [2]. This means that given an environment the values (e.g., communication, simplicity, feedback, or courage) must be interpreted to achieve a set of goals (e.g., quality), which is fixed. Ultimately we would like to gain as much flexibility as possible to widen the range of interpretations we can give to the values.

As we have seen, large companies provide a less then ideal environment for practicing the values, especially feedback. Based on our experience we describe in idiomatic form how to deal with such adversity.

3.1. Do As The Romans Do

Follow the prescribed procedures while staying faithful to your values.

3.1.1. Motivation

I spent my vacation on Australia's Surf Coast recently. Many of the beaches we visited had signs put up to warn you before strong currents. If you were caught by the "rip" and dragged off shore, so went the explanation, you should not try to fight against it but swim sideways. The rationale behind this is that a human being is way to weak for being successful against the current. Therefore, it is wise to hope for stiller waters at the side and be able to swim back to the shore there.

A large organisation is just too big and you are not likely to be in charge of everything. It resists change for a host of reasons like culture, investment protection, politics, etc. Trying to change the overall setup means years of work, not days. Therefore, for the duration of your project you can assume it to remain stable. Following the procedures is only sane if you want to optimize your chances of survival.

3.1.2. Forces

- Procedures try to achieve a certain goal; usually this goal is also your own, only the means are different.
- Following meaningless procedures is not only unhealthy; it can even be unprofessional.
- You have some leeway in interpreting the procedures so that they make more sense to you.

3.1.3. Implementation

Try to follow the defined procedures as close as possible. Use the language you are expected to use. Do not try to be smart and tell other people what is reasonable. Adapt your process to the predefined one, not the other way round.

3.1.4. Example

In our project we were obliged to write test scripts. It is used by end users during acceptance testing to make sure the software complies with the formulated business requirements. The data staging tool we integrated into metadata management was a black box. We could not provide test data for the end users to check the results for sanity. This argument did not meet with sympathy by the change control people. However, they accepted the following procedure: test results were not checked for absolute values, but the tests were based on consistency rules: if a data staging session execution failed, the archive of session executions had to contain this particular execution, etc.

3.1.5. Relations

Even if you Do As the Romans Do, you are still free to Explore the Limits of the environment you work in.

3.2. Demand Precision or Be Vague

Insist on clearly defined requirements or do not commit to scope.

3.2.1. Motivation

Imprecise requirements usually go along with stories of grand scope, and large stories are difficult to estimate. Committing to such stories puts you in jeopardy, because you promise to deliver something you do not understand. Oftentimes the customer does not understand such requirements, either. The normal thing to happen is feature creep, i.e. the effort rises while available time remains stable.

Another reason for such behavior might be an unrealistic schedule, mostly due to time pressure. In such a situation it is best for you to try to save yourself and hedge your bets.

3.2.2. Forces

- Large enterprises are the archetypical environment for projects of grand scale and unclearly specified requirements.
- Imprecise requirements give way to feature creep, which can jeopardize your release plan.
- Customers are usually imprecise; demanding full precision does not often meet with sympathy.
- Being vague allows you to change your mind late.
- Customers usually demand you commit to scope.

3.2.3. Implementation

Make the stories as small as possible. Try to make sure the customer is readily available for clarifications. Move large stories to the next iteration or release. If you absolutely cannot avoid committing to a requirements not understood, do not commit when and to what detail you will deliver.

3.2.4. Example

When we talked to business representatives what data and services they were interested in, the issue of active notification (push technology) ranked very high on the agenda.

3.2.5. Relations

Since you want to Do As the Romans Do it is less wise to fight unrealistic schedules than to commit in vague terms.

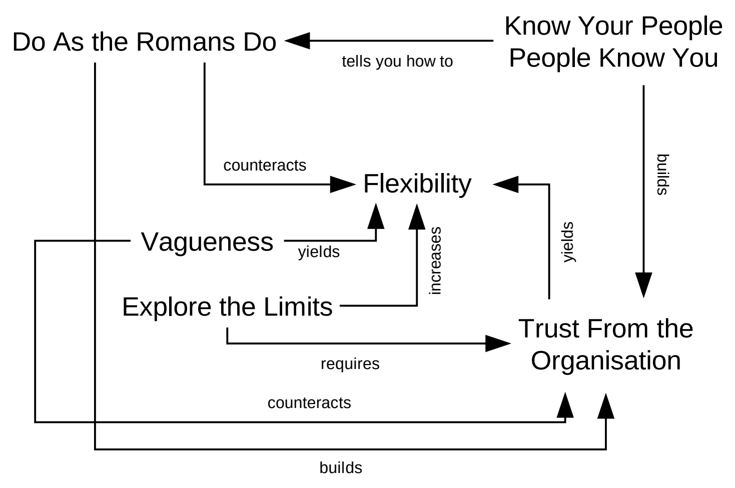

Figure 1: Relationships between practices easing and inhibiting the transition to XP in a large organisation. The ultimate goal is flexibility, but getting there might require a sensitive tuning of the forces at work.

3.3. *Explore the Limits*

Try to go as far as you can, but not farther.

3.3.1. Motivation

The organisation puts limits on what you can do. You cannot expect to have total freedom, but some degree of liberty is necessary for you to do a decent job. While some boundaries are set, others are not clearly defined and you might be able to find a niche to practice what you want to.

3.3.2. Forces

- You are unsure about how far you can go in your organisation.
- You might discover that the limits are too tight to achieve a reasonable amount of flexibility.
- The more Trust From the Organisation you receive, the farther they will allow you to go.

3.3.3. Implementation

Explore your limits carefully, step by step. Stay in contact with the people who might

3.3.4. Example

For every release we bring into production we must write an operation handbook. It is used by production support to understand how to deal with day-to-day problems like server crashes etc.

The handbook template alone is some 30 pages in size. It contained numerous sections that did not apply to our situation. It is very time-consuming to write it and takes away valuable time you would otherwise use for coding and testing.

We had to find out how much detail was required for the handbook to be accepted by change control. After the third attempt we were able to fine-tune the level of detail so that acceptance was almost certain.

3.3.5. Relations

While exploring your limits, be careful not to abuse the Trust From the Organisation you receive.

3.4. *Know Your People, People Know You*

Learn about their goals of people you depend upon and let them know about yours.

3.4.1. Motivation

In a large organisation many things happen behind the scenes. Unless you have been born in this environment, you will not overlook (let alone understand) all processes going on. Therefore your own point of view will be unrealistic, and if you are unrealistic you will not act according to reality. This might one day lead to clashes that cost time and money.

3.4.2. Forces

- The organisation is big and you cannot expect to fully understand it.
- You have to know the right people to understand the processes properly.
- Most people are trying to be helpful at first encounter, but some are not.

- Most people expect you are familiar with the procedures, but you are very likely to not be.

3.4.3. *Implementation*

Find out who is important in the organisation or has a good knowledge about it. If you run into problems, try to contact these people as early as possible and describe your problems to them. They will most probably also describe theirs to you and try to help.

3.4.4. *Example*

Each time we contacted people in the architecture department and asked questions, they were happy to answer them. Apparently it is uncommon for people to ask for advice in large organisations, be it because of lack of knowledge or lack of appreciation.

One day we were faced with a seemingly simple problem: adding the network address of database to a configuration file to make it available to our tool, just as other databases had been made available. We went to ask production support, but they would not grant us access to our configuration files. They said we would have to file a change (and go through the whole acceptance test procedure again).

On that day another colleague from production support happened to have a meeting with me. I asked him: "What would you recommend?" To my biggest surprise he mentioned that there is something known as a "Informational Change". This type of change is only filed so that production support knows what happened in case problems occur. Otherwise they let you apply the change without further tests. We happily filed the informational change request, and one day later we could apply it.

3.4.5. *Relations*

When you Know Your People and People Know You, it is much easier for others to gain trust in you. It also becomes easier to Explore the Limits.

3.5. *Trust From the Organisation*

Make people trust you.

3.5.1. *Motivation*

If people don’t know you, they won’t trust you. They will play not to lose instead of playing to win.

3.5.2. *Forces*

- The larger the organisation, the more people have to know you.
- People will usually not approach you to help.
- People appreciate friendliness.

3.5.3. *Implementation*

Actively search close contact with the people you depend upon. Ask them if what you do is alright in their eyes. Let them review your work. Let them know you appreciate their work. Say: "Thank You."

3.5.4. *Example*

We celebrate release parties not only among the team, but also invite colleagues from other departments who took part in the effort. During these parties people get to know each other personally. In our experience you are treated completely different by someone who knows you personally than by someone who only knows your mail address and phone number. As a sign of our appreciation, we pay for everything they consume at the party.

3.5.5. *Relations*

When you receive Trust From the Organisation, flexibility rises substantially and you can Explore the Limits easier.

4. *Outlook*

It should be pointed out that these recommendations are, after all, based on anecdotal evidence. However, we are pretty sure they will not make your life harder. They worked well for us, and it is assumed they can be applied in other large organisations as well.

We have just begun to transition to the complete set of extreme practices, and we are very careful in adding more practices as our project continues. The experience made so far suggests that it is possible to establish a flexible software development process in a hostile environment, but it takes its time.

References

1. Kent Beck: Embrace Change. Extreme Programming Explained. Reading 1999 (Addison-Wesley)
2. Dirk Riehle: A Comparison of the Value Systems of Adaptive Software Development and Extreme Programming. In Proceedings of XP 2000, Cagliari, Italy