1 XP—Risk: The basic Problem

SW development has a lot of risk. Schedule slips, project cancelation, sour system (high cost of manteinance), high defect rate so system is not used, bussiness not correctly understood so sw does not solve issues, bussiness changes and system cannot be updated, programmers leave, etc.

1.1 Solving these problems with XP

- **Schedule slips**: Short release cycles. Within release, 1 to 4 week iterations of features, one to three day tasks, high priority first.
- **Project canceled** Only do the smallest release first that makes sense in the bussiness so there is less to be bad and SW has more value
- **Sour System**: Comprehensive suite of tests.
- **Defect Rate**: customers write tests as well
- **Bussiness misunderstood**: Cannot happen if bussiness is part of the development.
- **Bussiness changes**: short release cycle, customers can substitute new functionalities.
- **Staff turnover**: we try to keep them with us, good feedback, programmers make estimates and it is not done by people outside the project. New members are gradually inserted into the team

1.2 Development Episode in XP

Basically, it is this:

You take a card from your stack of task cards. You ask another member for help. He has to say yes because he has. You write a test case, many test cases. Then you write the code of the task. Finally, if the code passes the test cases, it is integrated.

1.3 Economics of SW development

Three factors:

- Cashflows in and out

- Interest rates of the cashflow, we can get the net present value of cashflow
- Project Mortality: Multiply discounter cashflow by project mortality probability.

Knowing those, we can try to create a strategy to earn more, with superior marketing and sales organization, spend more later and earning sooner, so we have more interest on the money we receive, and try to reduce project mortality.

At each time, in a project, you have four options

- **Abandon**: You get the value from the originally envisioned form
- **Switch**: The project has more value if the customer can change the requirements halfway through the project.
- **Defer**: Wait until the situation sorts itself out, investing later without losing the project (doubts here)
- **Grow**: Grow quickly if the market is taking off.

There are five factors involved in the worth of these options. The **amount of investment** to get the option, the **price** at which the prize can be purchased if we use the option, the **current value** of the prize, the amount of **time** in which you can use the option (how much do you have to wait to get there, very related to 1), and the **uncertainty** of the eventual value of the prize.

The example of page 21 of XP by Kent Beck is gold. Read it if you forget it.

1.4 Four variables to control

Essentially, we have four variables. **Cost, Time, Quality and Scope**. The external forces choose three, and the team decides on the fourth. If the external forces do not like the result, they have to change it.

Essentially, you do not want to change quality, except perhaps by not testing, but the cost is to big. More time to deliver increase quality and scope, but too much time will reduce feedback. Too much money is bad, to little is super bad. Less scope will allow to deliver sooner, cheaper and with better quality, if the scope is too reduced it will not solve the problem.

1.5 Cost of change

Essentialy, cost of change cannot be exponential if we are using XP. Using OOP improves a lot only if it is done correctly. Object databases would help a lot as well. Of course, XP can be applied without OOP.

Simple design, no extra element, automated tests and a lot of practice in modifying the design, allow for a low cost of change, or at least, not too high.

1.6 Four Values

- Communication: Almost all problems in SW Dev are related to bad communication. Not asking the right questions leads to bad requirements, misreported progress, not reporting critical changes. Unit Testing, Pair Programming and Task estimation help people to communicate.

- Simplicity: Essentialy, do not do something unless it is needed. Do not overcomplicate stuff. Make it work first.

- Feedback : You have small feedback, like unit testing, estimation of customer stories and the progress tracker, and big feedback, like functional tests for all the stories, like use cases, and the customers review the schedule of the project every three weeks aprox to make sure that the velocity is fine. Also, early production generates concrete feedback, because the system is actually tested.

- Courage: is to go fast, to be able to change big things when we have advanced a lot, to be able to throw code.

1.7 Basic Principles of XP

- **Rapid feedback**: The feedback needs to be as inmediate as possible.

- **Assume simplicity**: Aim to build everything as simple as possible. Then try to build up in complexity but only when it is needed

- **Incremental change**: No need to explain this. Think man!

- **Embracing change**: The best strategy it the one that preserves the most options while actually solving your most pressing problem.

- **Quality Work**: You can do excellent and insanely excelent.

Other stuff to keep into account: Teach learning, small initial investment, play to win, concrete experiments, honest communication, work with people instincts, accept responsability, local adaptation, travel light and honest measurement.