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.

2 Clase 23-03-2020

2.1 Second P: Product

Final result of the development process. To define it, we must define the 'ambito'. It is the context, the information, its function and the 'rendimiento' that it must have. Security, usability, goes here too.

SW must be non ambiguous, its 'ambito'. We must meet the client and check and recheck our 'ambito'. We must keep it short. If a functionality is not necessary, we must not implement it. Divide and Conquer!

We must define the limitations and restrictions that the SW must have. This is what allows us to face Risks.

2.2 Third P: Process

It defines what we are going to do. Processes and practises, like Agile, must be modified to work in your environment. Do not apply as is.

The process influences in how we develop our product. The process influences how we manage our project.

2.2.1 A development episode

Go from a task related to a feature, make a test, then a implementation, given the design.

We need more valuable sw, waste money slowly, win more money and win more viability.

The four variables. Cost, Time, Quality and Scope. Focus on scope.

To be able to change, we need a lot of stuff, including regression tests.

Basic Activities:

- Code: At the end of the day, there has to be a program.
- Testing: Only things that can be measured, exist. TDD.
- Listen: Because we dont know shit
- Design: Architecture that can scale and work. x

3 Cap 1: User Story

It describes functionality that will be valuable to either a user or purchaser of a system or sw. They have three things: **Written Description, conversations about the story and tests to check if we finished**

These are usually called Card, Conversation and Confirmation. Customers must be able to value Stories and understand them.

Stories that are too long are an epic. Epics must be splitted into smaller stories.

Stories can include expectations that can be used to test your story. The customer team writes the story cards. The stories are prioritized by the organization. Each story has story points. Releases and iterations are planned by placing these stories into iterations.

Velocity is how fast the team works.

User Stories are better than use cases because they emphasize verbal communication, can be understood by the client and they encourage the deferring of detail

4

5 Class 26/03

Version one: Empresa EEUU. Se dedica a recabar informacion sobre Agile

Agile used 97 percent in organizations

Percentage of teams using agile: 22 percent say that all their teams are agile. 26 percentage of their teams are agile.

Generally speaking, most teams are using Agile instead of traditional ones.

1968: NATO. Do we work in SW as common ingeneering? 1970: Waterfall 1985: DoD recommends waterfall 1994: Crystal, ADAD 1995: DSDM, Scrum 1999: Xtreme Programming.

Agile Manifesto: People above process Working Soft: Less Documentation Colaboration with the client Change over following a plan

Projects must be build around motivated individuals

Face to face dialog is the most efficient method. Osmotic communication.

The only way to check if there is progress, is working sw.

Agile processes promote a sustentable development

Continous attention to technical quality and good design improves agility.

KISS: la simplicidad es lo esencial.

Self organized teams.

Look for authority matrix.
The team gathers and reflects on how to be more effective.

6 Writing Stories

Stories are **invest**

- Independent: Stories must not depend on each other
- Negotiable: Each card can be changed or discarted
- Valuable: Stories must be valuable for users, not just for developers. If it is possible, the customers must write the stories.
- Estimatable: The issues may be that the developers lack techinical knowledge, or technical knowledge, or the story is too big. When there is a lack of technical knowledge, you can do a **Spike**, a small training exercise where you can get some knowledge, enough to do a estimation. If the story is too big, it needs to be sliced into smaller stories or to be made an epic or pospone the estimation, or make a very general one.
- Small: Essencially, if your story is an epic, you have to split it. There are two kinds: **Compound Story**, where the epic is comprised by multiple shorter stories. The other case is the **Complex Story** where the story cannot be disaggregated. Sometimes it is just because we do not know enough, so we have to make a spike. In that case, it is better to put the spike as a separate story and estimate it.

  If the story is too small, like small UI changes, it is better to combine a few into a single story.
- Testable: Sometimes, nonfunctional requirements are not easily testable. Whenever possible, tests must be automated. Stories must be well written in order to be testable. If the story says 'The sw must always work' is not testable at all. On the other hand 'The sw must not crash 99 percent at start' then, even though not necesarilly a good story, that is testable.
-

7 Clase Martes 30/03

Como maximo, tres semanas o cuatro necesito un incremento, como minimo esuna semana, ciclo iterativo o incremental, con equipos autoorganizados.

Projecto Agil: ciclo iterativo e incremental con equipos autoorganizados

An agile team must be small: Beyond 9 people, adding more people is not productive.

What is the problem, when we talk about a bigger team? It is difficult to reach consensus.

The number of links in a group is important. Given n people in a group, n*(n-1)/2 is the number of links. According to scrum we also need more than 3.

The team must be multifunctional. First we define tech necesities, and then we need at least one expert in each tech. And then, we need at least someone that knows a bit of the tech as well. If the expert has an issue, the guy who knows a bit, can step up, but then it will have less quality, so then the expert can look it and get it better.

Skill matrix here. Stars and dots.

It is not excellent, if you have to share a lot a certain person among a lot of groups, but it can be done.

An Agile Team must be self-organized: The guys from above say what we must do, and the team says how will we do it. The team must also be adaptive, be able to change if there is any issue.

Self organizing can have a dark side: Sometimes people do not do stuff because everyone is responsable, or even worse, someone not capable takes responsability because there is an emergency.

Self organizing is not anarchy. There are bosses. The team must answer to somebody. To solve responsability issues, we can use delegation boards, rows are activities, and columns are levels of delegation.

The levels are a rule but the people in charge, down here, are just examples:

First, second and third level are bosses' activities. First level is **tell** The boss takes decisions and communicates them. Selecting a team is at this level.

Sell is the second level. The scope for each release is at this level. Usually done by who?

Consult take decisions according to consultations with members of the team.

Agree: team makes decisions and then ask boss for agreement.

Advise: the boss just gives advice, the team can use the adivce or not

Inquire: the team asks the boss about certain decisions that must be taken.

Delegate: the decisions are delegated to the team.

We can put the person who is in charge of the task, in the intersection in the delegation board. Not mandatory.

8 Class 2/4/20

Definition of User Stories. Smallest Unit valued by the bussiness. It is common that User Stories have the following structure:[Title] As [Rol], I want to [functionality here] with the goal to [Benefit]

A User story must be started and finished in a single sprint.The story must include the acceptance criteria. Those acceptance criteria will also help to determine tests.

Defining the story users is responsability of the product owner, the client. The Scrum master can help, for example, to the product owner. The client must also prioritize the user stories. Then, the stories must be negotiated.

Parts of a user story: Card, conversation, confirmation. User stories can have mock ups. For certain projects, when documentation is mandatory, we make documentation as late as possible.

Long stories, epics, will be fractioned. Epics will go into story maps, or roadmaps as well. Each User Story must be divided in tasks, where the how is defined.

In the product backlog, just user stories. In the sprint backlog, there goes the tasks of the user stories.

Tema: Theme, bigger 'not knowing' about the functionality. Like modules: facturation module. In the buying module, we have the epics: add provider, new buy, etc. For each Epic, we have User Stories, for each User stories we have tasks

The user stories are validated in the sprint planning. User Stories can be obtained all the time

The product owner brings a lot of stuff, that goes into the sprint backlog. The sprint starts, during that time, the product owner can discover new needs. those new needs are added to the product backlog with new user stories. At the end of the sprint, we deliver the increment, the working sw. We take the new stories and redefine the sprint backlog. The user keeps on getting new needs, playing with the working sw. It is called **Continous Discovery - Dual Track**

The product owner and scrum master can meet during sprints, to plan next sprints, specially because of the discovery of new needs

The INVEST Method is used to check the quality of a User Story.

Stories must not be codependent. Must be negotiable. You can debate about stories. They must have value for the user They must be estimateable, in user points