Chapter 27

The XP Game Explained

—Vera Peeters and Pascal Van Cauwenberghe

Part III

The XP planning game, velocity, and user stories are hard to explain and "sell" to developers and businesspeople. How can we better explain these concepts? The XP specification, estimation, and planning methods sound too weird and simple to work. How can we prove or show that they work, without committing ourselves to "just do it" on a real project? If the project is to succeed, we need to get developers and businesspeople working as one team. How can we get them to talk, cooperate, and trust each other?

The "XP Game" is a fun and playful simulation of the XP development process. No technical knowledge or skill is required to participate, so we can form teams with both developers and businesspeople. The element of competition helps the players bond with their teammates. At the end of the game, everybody has experienced how user stories, estimation, planning, implementation, and functional tests are used. They have seen how the "velocity" factor is used to adjust the schedule. Developers and customers have gotten to know and respect each other.

Introduction

Lesire Software Engineering had been using "developer-only Extreme Programming" for a few months. Refactoring, unit tests, simple design, and pair programming had resulted in measurable improvements in the software process.

When the company wanted to transition to "full" XP and expand the use of XP beyond the development team, it faced a number of problems.

- Poor communication between business and developer teams.
- No understanding of user stories, the planning game, and velocity.
- Distrust of the planning game and velocity. How could such simple methods deliver credible plans and schedules?

Presentations, discussions, training, and coaching did not completely resolve these issues. Developers and businesspeople reluctantly agreed to try the XP experiment, without much hope of success.

Then Vera Peeters, Lesire's XP coach, exclaimed, "Let's play a game!"

Let's Play!

The players are divided into small teams (four to eight players). Each team consists of developers and customers. A coach assists each team, to explain and guide the game and to answer questions. The team can earn "business points" by performing simple tasks. The team with the most business points wins.

The coach gives the team a small set of prewritten story cards. These cards describe simple tasks, such as "Build a two-story house of cards," "Throw a six five times using two dice," and "Find a missing card from a pack of cards."

The team members (acting as developers) estimate how long it will take them to "implement" the tasks. The coach is available to answer questions about the stories. The team may choose a time between ten and 60 seconds. Or they may declare that it's impossible to implement the task in 60 seconds. When all the stories have been estimated, the cards are handed back to the coach.

The team (now acting as a customer) must create a release plan: They choose the stories to implement and the order of implementation. Each story is worth some business points. The team tries to maximize the number of business points they can earn. The total time to implement all selected stories may not exceed 180 seconds, the fixed iteration time.

The team members (acting as developers) must now implement each planned story in turn, in the order defined by the customer. An hourglass is used to measure the implementation time. When the implementation is "accepted" by the coach, the team earns the business points of the story. When the hourglass runs out, the iteration ends. If the team has finished all the stories before the end of the iteration, they may ask the customer for another story.

At the end of each iteration, there is a debriefing, discussing the problems, solutions, and strategies. The "team velocity" is explained and calculated. For the next iteration, the team must use velocity instead of time as a measure of how much work they can do.

The simulation typically runs over three iterations. It takes from one and a half to two hours, including debriefing and discussion sessions.

Open the Magic Bag

At the start of the game, a bag with game props is emptied on each team's table. The bag contains dice, playing cards, colorful paper, balloons, pieces of string, folded hats and boats, story cards, planning/scoring sheets, a pen, and an hourglass.

This jumble of colorful, childish props reassures the players that this is not a serious event. They will not be judged; they can relax and enjoy the game. They are open to bonding with their teammates and to absorbing something new. It's just a game, after all.

Tell Us a Story

The story cards contain simple and small tasks such as "Find a missing card from a deck" or "Inflate five balloons to a size of 40 cm." No special knowledge is required to understand how to carry out these tasks.

And yet, the cards alone do not contain enough information to estimate the complexity of the tasks. The players ask the coach (acting as their customer) more questions about the tasks—for example: "Are we allowed to perform the task as a team?" "What do you mean by throw a six five times? Do we need five consecutive sixes?" The coach clarifies the meaning of the stories and explains what conditions must be fulfilled to accept the story.

This reinforces the idea that user stories aren't specifications but "a promise to talk." It's all right to ask the customer questions. Get as much information as you need to make good decisions. Ask the customer—they know! Or at least they can find out the answer.

Estimating: How Hard Can This Be?

One of the most difficult aspects of working with velocity is convincing developers to estimate consistently. The velocity factor includes all the variable parts of team performance: experience, knowledge of the problem domain, team size, adding and removing team members, interruptions, and so on. If the planning game is to result in realistic schedules, stories should be estimated consistently in terms of their relative required effort. Developers should not try to adjust their estimates.

The coach asks the players to estimate by comparing with other stories. If a story looks twice as difficult as another story, estimate it at twice the difficulty. The mix of stories proposed in the documentation contains stories that are suited to demonstrating this. For example, folding eight boats will probably take twice as long as folding four boats. There are also several stories that have the same complexity. For example, throwing a six five times is obviously exactly as difficult as throwing a four five times.

But it's not always that easy. Building a two-story house of cards is more than twice as difficult as building a one-story house. And finding two missing cards from a full deck is about as difficult as finding one, if you sort the cards.

Estimating how long it takes to perform these little tasks is difficult. Developers already know estimating is difficult. For some customers, it might come as a surprise to learn how hard it is.

One way to improve the estimates is to do short experiments, called "spikes." You could implement these simple stories completely as a

spike. With real stories, this is not possible, so we give the players only a limited time (for example, 15 minutes) to come up with their estimates.

Insanely Short Iterations

XP recommends short iterations, so the XP Game uses *really* short iterations: 180 seconds. These 180 seconds include only the "pure" implementation time, without counting any preparations or the time to perform acceptance tests.

These 180 seconds are the "budget" the players can allocate when making their release plan: The total estimated time of all the chosen stories must not exceed 180 seconds.

If all the chosen stories have been implemented and there's time left, the customer may choose some more stories. The iteration ends after exactly 180 seconds, not a second more, not a second less. If the team is still working on a story, this story is abandoned. This emphasizes the fixed iteration time, during which the customer can change only scope and priority.

When the iteration is about halfway through, the team compares their actual implementation to their plan: Are we ahead, behind, or right on schedule? Should we warn the customer that they might have to reduce scope?

We used to measure time with a stopwatch, but an hourglass is more fun, more tactile, and more visible. We don't care about one-second precision; we need to know if we're halfway or almost done. It turns out that even with such simple tracking tools as a list of story cards and an hourglass, we can give the customer pretty accurate and useful feedback on our progress. The sight of the last grains of sand sliding down the hourglass reminds the players that time is finite and increases the pressure to finish the implementation of their story.

Planning with Risk

Most of the stories depend only on skill and teamwork. We can expect the team to perform consistently on these tasks.

But some stories depend on luck. How can you estimate how long the dice stories (for example, "Throw three ones with two dice") will take? You can compute the odds; multiply by the time it takes, on average, to

throw the dice. The answer might be "On average, it will take about 30 seconds to throw three ones with two dice."

When you're planning, you need to take this uncertainty and risk into account. If the estimate is 30 seconds, it might as well take ten or 60 seconds, depending on your luck. When you have two stories that will earn you about the same business value, but one is more risky than the other, which one do you choose? Some people choose riskier stories with a higher payoff; some people prefer the stories with a lower risk. The important thing is to explicitly take the risk into account when planning.

Customers learn that it's not always possible to come up with definite estimates. Developers learn that it's hard to plan when story estimates include uncertainty.

Silly Little Games

The stories are silly little games: throwing dice, sorting cards, finding missing cards, building a house of cards, folding a paper hat, inflating balloons. You can invent your own additions, but all these stories have some features in common.

- They add to the relaxed, playful atmosphere.
- They are not a bit technical, so everyone on the team can join in and contribute.
- They are difficult enough to require some concentration and team effort to complete quickly.
- They are not too difficult, so the players can concentrate on the XP concepts we want them to experience and learn.

The team decides before each game how they will perform the task (in team or solo) and who will perform the task. Players sign up for tasks; tasks are not assigned to them.

All the planned games are played one at a time, in the order chosen by the customer. This reinforces the idea that the whole team is responsible for all the stories.

It's fun to see grown men and women working so hard to fold paper hats or build houses of cards—or design and execute a six-way parallel sort of a deck of cards!

Playing to Win

The element of competition is important to the game. Everybody in the team cooperates to implement as many stories as possible, as fast as possible. They want their team to earn more points than the other teams. Putting developers and businesspeople in one team makes them work together, maybe for the first time.

Don't create teams with only developers or only businesspeople. We don't want to see who's "smartest"—we want to foster cooperation.

Acceptance Testing: Don't Trust Developers

Most of the questions the players ask about the stories are related to the acceptance criteria of the story. The coach (acting as the customer) tells the players how he will test whether a story is implemented correctly. For the balloons, there are pieces of string to measure whether the balloon is big enough. For the paper-folding stories, there are prototypes of the boat and hat to be folded.

Some stories look so easy that no explanation is necessary. But what do we mean by a "sorted deck of cards?" Is the ace the first or the last card? Do we need to put the different suits in some order? The team has to agree with the customer/coach up front to estimate the difficulty of the story.

When the team has implemented a story, the coach explicitly performs the acceptance test: The sorted deck is verified; the folded hats are compared with the prototype; the balloons are measured with the piece of string. When the implementation doesn't pass the test, the team must continue the implementation. Even the simplest tasks can be implemented incorrectly, so an acceptance test failure underscores the importance of testing.

Trading Places

During the game, the whole team switches between developer and customer roles. The coach emphasizes this fact by displaying a banner, which reads, "We are the customer" or "We are the developers." If the players must switch roles, even briefly (for example, when they ask for more stories from the customers), the coach explicitly makes them switch roles.

This makes it clear that the customer and developer roles are distinct. Each shares some part of the responsibility to succeed. Both have well-defined tasks. Developers should never make customer decisions; customers should never make developer decisions.

Switching roles lets everybody experience all aspects of the cooperative game of software development. Developers and customers experience what "the other side" does during a project. This increases the respect each group has for the work of the others.

Velocity: How Much Did We Get Done?

This is the trickiest part of the simulation: explaining the concept of velocity after the first iteration.

After each iteration, each team announces the number of business points earned and the sum of the *estimated* durations of the implemented stories. These numbers are easily found on the planning sheets. Let's call the sum of the estimated durations *X*. Note that we don't know the actual duration of each story, because we use an hourglass.

What we want to know is, how many stories can we implement in one (180-second) iteration? Well, we just did such an iteration. How many stories did we implement? We implemented some stories requiring *X* amount of work. How much work will we be able to perform next time? The *Yesterday's Weather* rule says, probably about the same amount as in the previous iteration. That's if we don't take sick leave, holidays, interruptions, team learning, reuse, refactoring, and a thousand other details into account.

Let's give *X* a name: "Velocity." Velocity measures how much work the team can do per iteration. If we want to finish the next iteration on time, each team will schedule its Velocity's worth of stories.

If Velocity is effort per iteration, the story estimates don't represent time, but estimated effort. We no longer estimate in seconds, but in "points" or whatever you like as a unit of programming effort. How can we estimate using this made-up unit system? We estimate by comparing with other stories: If a new story is about as difficult as an implemented story, give it the same number of points. If it looks twice as difficult, give it twice the points. These points really express the relative complexity of the stories. Maybe a good name would be "complexity points." The important thing is to estimate consistently.

What about all these pesky factors that influence how much work the team can do? We let the velocity take care of them. If the team learns about the domain or reuses more code, they get more done. Their measured velocity rises; they take on more work. If new members join the team or people go on holiday, the measured velocity decreases, and the team takes on less work. It really is that simple! The most difficult thing is *not* to take all those factors into account when estimating.

Does this really work? In the simulation, the teams typically underestimate the stories in the first iteration and end with a velocity lower than 180, which indicates that few teams can estimate even these simple stories accurately. In the second iteration, they get to ask for more stories (because they get better at doing the implementations) and increase their velocity. By the third iteration, they get pretty close. In real life, the authors have experienced that the velocity stabilizes after only a few iterations. This has been observed in teams that worked for several iterations on the same project and in teams working on different projects in the same domain, using the same tools. Expect your velocity to change dramatically if you change your team, problem domain, tools, or environment.

Sustainable Playing

Each iteration has a fixed size of 180 seconds. Before implementing each story, the players devise their strategy. The hourglass is started when the implementation starts. When the team thinks it has completed the story, the hourglass is stopped. The hourglass only measures real implementation time.

The implementation time is very stressful, because the players try to complete a task in only a few tens of seconds. The players can relax a bit between implementations and discuss their design and strategy. This simulates the XP practice of sustainable pace or "no overtime": Developers work with full concentration on some task, go home, relax, and come back the next day, ready to perform another task.

Despite the concentration and stress of the story implementations, the players don't feel tired after three iterations. That's because they enjoy what they're doing, and they get time to relax between tasks.

Conclusion

The XP Game is a fun and playful way of simulating the XP process. It enables the participants to experience, in a relaxed environment, several of the XP practices: user stories, estimation, the planning game, acceptance tests, short releases, sustainable pace, progress tracking, and velocity. It is ideally suited as a follow-up to a tutorial on these practices. The competition, team effort, and fun aspects make it a great team-building tool: Playing the game improves trust and communication between developers and businesspeople.

The XP Game has been played in companies transitioning to XP, during XP tutorials, and at software engineering conferences. The participants always have fun and feel they have learned more about the XP practices.

And Lesire Software Engineering? They're still doing XP and delivering working software to their customers. Developers and businesspeople are working together and enjoying it.

References

Au: Pls. add [Beck2000] after actual citation within chapter text.

[Beck2000] K. Beck. *Extreme Programming Explained*. Addison-Wesley, 1999.

Acknowledgments

We thank all those who played and coached the XP Game for helping us improve the game and for the fun we had doing the game.

We thank Gerard Meszaros for making us think about what we were trying to teach.

We would like to thank Lesire Software Engineering and the XP2001 and XP Universe conferences for letting us play.

About the Authors

Vera Peeters is an independent consultant with more than ten years of experience in implementing object-oriented systems in a variety of domains. For the last two years she has been working with XP, helping her customers transition to XP, coaching XP teams, and giving lectures and

presentations about XP. Vera is involved in the organization of the Dutch and Belgian XP user groups.

Pascal Van Cauwenberghe is the CTO of Lesire Software Engineering, where he leads the software development teams. He has been designing and building object-oriented systems for more than ten years.

The XP Game is free and available from http://www.xp.be.