NAYIMA TrYx

XP Game

The Manual

By Vera Peeters and Pascal Van Cauwenberghe

Version 5.0. June 2008.

See http://www.xp.be for updates and further material.

We hope you enjoy playing the game!

Vera Peeters
vera.peeters@tryx.com
Tryx
http://www.tryx.com

Pascal Van Cauwenberghe
pascal@nayima.be
NAYIMA
http://www.nayima.be

What is the "XP Game"?

The XP Game is a simple game, which lets customers and developers experience and learn…

How the XP planning game is performed to negotiate an iteration plan.

How velocity is measured and used to make a predictable schedule.

How developers and customers play their parts in the planning game.

The game is typically played in an organization that wants to start implementing the XP planning game. The participants should know about XP, but no experience is required. The game is played with a few small teams. Each team consists of developers and customers. An experienced coach assists each team.

The teams go through at least three XP "iterations". In each iteration the team performs a planning game session (based on pre-written story cards) and an "implementation" phase. Each correctly implemented story earns the team some "business value". The team with the largest earned business value wins.

Team velocity is measured after each iteration and then used to plan the next iteration. Ideally, the teams should be able to plan correctly after a few iterations.

Game requirements

Players

No technical knowledge or skills are required. No XP experience is required.

Teams should be composed of 4 to 6 customers **and** developers, to promote cooperation and communication.

A team plays the role of "customer" in some parts of the game, and the role of "developers" in other parts of the game. We use the terms "customer" and "developer" here to describe 2 roles in the XP process. Roughly, the customer takes business decisions, the developer takes technical decisions.

Each team requires a coach. If there are more teams than coaches, some coaches may have to assist more than one team. Only experienced XP Game coaches should coach more than 1 team. Be sure to point out that not having access to a full-time coach (or an "onsite customer") will lead to inefficiencies, as the players will lose time waiting for their coach to become available.

Coach

The coach explains the game to the team. The coach should have experience with XP and the XP Game.

The coach handles the hourglass and writes down the scores on the Scoring Sheet. If a coach has to assist more than 1 team, these tasks can be done by a volunteer in the team. This person will not help with the implementations. It's best if another person does this in the next iterations, to give everyone the opportunity to experience all phases of the game.

The coach answers questions about the game. The coach emphasizes which role the team is currently playing. He can use the Customer/Developer Responsibilities Sheets to do that. The coach represents the customer when the players are developers; the coach represents the developers when the players are customers.

The coach answers questions about XP. Typically, a lot of the questions center around estimating and velocity. This document contains an appendix with some common questions.

The coach helps the team estimate and plan.

The coach performs the "acceptance test" for each story, to verify if the players have "implemented" the feature correctly.

One of the coaches acts as "Game show host", introducing and explaining the different steps in the game.

Preferably, the coaches are briefed shortly before the session, to refresh the game rules and to agree on their roles. They also have to communicate with each other during the game, to make sure that different teams don't get different interpretations of some of the rules than others.

Terminology

Story: a short description of a feature that, when implemented, will provide some value to the company.

Acceptance Test: a test performed by the customer, to verify if the story has been implemented correctly.

Business Value: the "value" of the completion of a story to the customer.

Story points: measures how difficult it is to implement a story. (=how much time it will cost)

Duration

The game takes three hours for three iterations. This includes time for a debriefing and discussion session after each iteration. Allocate plenty of time for discussion.

> WHY?
>
> During the game, the participants are completely focused on the game. They don't have any time to reflect on what they're doing or why they're doing it. Explicitly take the time to discuss after each round what the participants have experienced, what they have learned from this and what they will do differently next round. This will let the participants reflect and learn, just like in a "Project Retrospective" [Kerth 2001].

A sample schedule for the game could be as follows:

Time (minutes)	Task
5	Introductions
20	Explain the concepts and the game
40	Iteration 1
25	Debriefing + introducing "velocity"
25	Iteration 2
15	Debriefing and discussion
25	Iteration 3 + celebrating the winning team
25	Final debriefing and discussion
180	

Preparation of the game room

For each team, you have to prepare 2 tables. One table is the "developer table", the other is the "planning table".

The participants sit around the developer table during the intro, and in the estimation and implementation phases of each iteration. In these phases, the participants play the role of developers.

The participants stand in front of the planning table during the planning phase of each iteration. In this phase, they play the role of customers. The planning tables can be put against a wall. They need no chairs.

> WHY?
>
> We want to teach the participants the responsibilities of the 2 roles: "developer" and "customer". The entire team of participants will switch roles. They can only learn this if they can feel the difference when they are in one the two roles.

You can find a detailed description of the materials you need for the game later in this document.

Game intro

The Game Show Host explains briefly what the participants can expect, and how the game will proceed.

You can use the provided presentation to do this. You can distribute the "Game Rules" document before you start the presentation. Don't spend too much time explaining all the details in advance. They won't understand it anyway, not at this moment. They will understand it after the first iteration. The important thing is that the participants know what to do.

Questions you can ask before the start of the game:

Which players are developers in real life? Which players are customers? Each team should have both of them.

Who knows about Planning Game and Velocity?

Who actually does XP in real life?

The participants group themselves in teams of 4-6 members. Each team sits around a table. Make sure that each team consists of real-life customers and developers. If you have participants that know about XP, distribute them over the teams, so that they can help to coach and answer questions about XP.

> WHY?
>
> We want to give the participants the opportunity to learn from each other. This works best if each team has members with mixed experiences and interests.

You don't have to create the teams in the beginning of the session; you can do this just before the game begins, when the presentation says "Enough Talk".

The game begins

After the intro, give each team a bag with game props.

Start the timer in the presentation. In the first iteration, the time allocated for the different phases is:

Estimation: 15 min

Planning: 5 min

Implementation: 20 min

The bag

These times are only guidelines, but encourage the team to work faster when you see they take more than the given time for a phase. We want to avoid spending too much time on estimation. We also want to avoid one team finishing much earlier or later than the others.

Empty the bag on the table. The team can have a look at the props.

Game iteration 1

Customer writes stories

Team members act as developers. They sit at the developer table, where the "Developer Responsibilities Sheet" is visible. Keep reminding them about these responsibilities.

The bag holds 3 sets of story cards. Pick the pack with story cards that is marked "iteration 1".

The first set contains 10 story cards. These cards contain the following:

- Short description of the story
- Business value from 100 to 500. The team must earn as much business value as possible. The business value is awarded when the story is "implemented" correctly.
- 1-2-3-4-5-6-impossible: choices for difficulty estimation. When estimating, the team should circle one of these options.

The team may *briefly* study the cards.

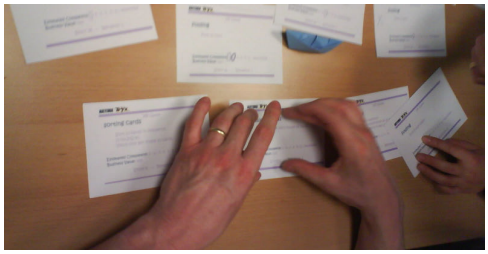
Developers estimate stories

Team members still act as developers. The coach answers questions about the stories.

We want estimates that are reasonably accurate but we don't want to spend too much time estimating. Each story card contains the numbers from 1 to 6 and "impossible". We express our estimate by circling one of the numbers (or "impossible"). We'll see next how we decide which number to choose and when to choose "impossible".

Step 1: order stories by effort

The players lay out all the story cards on the table, ordered by increasing implementation effort. Implementation effort means here: how much time it will cost to implement this story with the team.



Start with 2 stories: which one will take more time? Then add a third story: will it take more or less time than the first one? Will it take more or less time than the second one? Add all the stories one by one.

Put the easiest (shortest) stories at the top, the most difficult ones (longest) at the bottom. If two stories seem to require about the same effort, they are put next to each other. Check to make sure everyone in the team is happy with the ordering.

The team should ask the coach about the stories, acceptance criteria, how the story is to be implemented (e.g. can the whole team work on implementation or must one of the team members implement the story?) You can find tips about the particular stories later in this document.

> WHY?
>
> The stories on the cards don't specify these (very simple) tasks, there's simply not enough information. To be able to estimate accurately, the team must agree with the customer (role played by the coach) what the acceptance criteria of the task will be. Thus, we simulate the XP practice of agreeing on "Acceptance Tests" upfront with the "Onsite Customer".

Step 2: estimate the easiest story

The easiest story (the one at the top of the list) is assigned an effort score of **2** by definition. We indicate this by circling the number two. That was easy!

Step 3: estimate the other stories relative to each other

If the first story's estimated effort is 2, what is the effort required to implement the next story?

If the next story requires about the same effort, it's 2.

If the next story requires about twice as much effort, it's 4.

If the next story requires less than twice the effort, it's 3.

Do the same for each following story: compare it with the previous stories and assign the number of points accordingly.

At the end of the estimation session, the coach can ask the players to look at all the stories and see if the relative estimates look right.

> WHY?
>
> Later, when we introduce "velocity", we will see that it's not important to estimate correctly, only to estimate consistently. The coach should ask the team members to look over all the estimates and verify if they seem consistent. If not, the team can modify their estimate.

Impossible stories

If a story requires more than 3 times as much effort than the first one (thus, more than 6 points), choose "impossible". This story is too big to estimate correctly and implement quickly. Set this story aside to be split up into smaller stories by the customer (not done in the game).

> WHY?
>
> Some tasks on the cards are a lot more time consuming than the others. We prefer short stories in real life, as they're easier to estimate, give us more flexibility in planning and are less risky to implement.
>
> We've learned that you can almost always break up a large story in several small stories that deliver business value on their own. But only if you really try...

Step 4: how many stories can we implement?

The customer will want to know how many stories the team can implement in one iteration. An iteration in the XP Game is 180 seconds long (only counting the implementation time). How many stories can the team implement in 3 minutes? Hmmm... Hard to say. We've never done these tasks before. How can we guess accurately?

Well, we can't make an accurate estimate yet; we just don't have enough information. But don't worry: we'll soon know how much we can do in one iteration.

> WHY?
>
> How can you estimate how long something will take if you've never done it before and if you have nothing to compare it to?
>
> The best way to learn to make more accurate estimates is to perform some of the tasks and measure how long it took. That's what the first iteration is for: implement a few useful stories so that we know how much effort they take. And to give some useful functionality to the customer too, of course!

You, as an experienced XP Game coach, will be able to give the team an educated guess. Let's use 12 as our first guesstimate of how much work the team can perform per iteration. If you see that the team has mostly very low estimates, you may want to guess only 9 or 10 for them. If the team has many high estimates, you may want to guess 15.

Step 5: let the customer plan the stories

When all the story cards are estimated, the team members give the cards to the coach, who thanks the development team.

Customers choose and prioritise stories

Team members act as customers. They move to the planning table, where they will stand up. The "Customer Responsibilities Sheet" is visible at the planning table. Keep reminding the team about these responsibilities.

The job of the customer is to choose the stories and their order of implementation in the iteration. To do this, the customer takes into account the expected business value and estimated effort, which are both written on the story cards.

Step 1: order stories by business value/effort ratio

The team lays out all the stories, ordered by business value/effort. The stories with the highest value/effort ratio come first. For example: a story that is worth 300 points and costs 2 points comes before a story of 500 points costing 4 points.

Step 2: select the stories to implement

As customer, you would like all of these stories implemented, to get the highest possible business value. But developers can't do an infinite amount of work in one iteration. How much work can they do? Well, in the previous step, the developers have guessed how many points they can implement per iteration (probably 12 points). It can be different for each team. That's the "budget" you're allowed to spend in the next iteration.

The team must choose stories so that the total number of points for this iteration does not exceed the estimated points per iteration. For this they select the stories with the best possible business value/cost ratio. The idea is to "buy" as much value as possible, within the limited budget.

Step 3: choose the order of implementation

We now know which stories to implement. The team must order them for implementation. How do you decide on this ordering? Well, one thing to take into account is that the developers can only **guess** how many stories they can implement in this iteration. They could be wrong. They probably are. Stories that are scheduled early in the iteration are more likely to be delivered than those that are scheduled later. So, schedule the most important stories first. Which stories are most important? You decide: those that earn the most business value, those that require the least/the most time, those with the best value/effort ratio, those that are destined for the most important customer...

Schedule the stories in such a way that your chances of earning business value are greatest.

Step 4: make a public iteration plan

The result is an iteration plan. The developers will implement these stories in the order laid out by the customer. Each time the developers start with a new story, the coach writes down the story number, story description and estimate. When the story is implemented, the coach adds the earned business value. Thus, everyone can clearly follow the progress of the team.

> WHY?
>
> It's important to create visibility for the plan and the progress of the team. We could put the planning/tracking sheet on a big whiteboard or on an intranet page. Just make sure that the information is easily and frequently updated and that everybody can see all the relevant information.
>
> Such publicly visible information gives instant feedback to all stakeholders and allows them to intervene as soon as problems become visible.

Developers implement stories

Team members act as developers. They go back to the developer table.

The Coach acts as customer to answer questions and to perform acceptance tests. The Coach also acts as tracker to track story progress.

If each team has a full-time Coach, the Coach will also "guard the time". If the Coach assists more than 1 team, one of the team members will do this.

Executing the iteration plan

The team takes each story in turn, in the order defined by the plan. The complete iteration lasts until the hourglass is empty, simulating the use of "time-boxing", fixed-size iterations.

Only implementation time is measured. Time spent allocating tasks, preparing, performing acceptance tests, is not measured.

> WHY?
>
> We want to simulate "sustainable pace": periods of intense work alternate with periods of rest.
>
> To measure how much work we can do we only look at the time the team is productive, we don't measure how long other "overhead" tasks take. In real projects we will try to minimize the amount of time spent on overhead.

Implementation of a story

For each story, the team can briefly discuss the story implementation. Team members sign up for the task. The team can work together on most tasks.

Each story must be implemented completely. The team has to come to an agreement with the coach about the way they will let him know that the implementation is finished. The team determines when the story is finished.

WHY?

The team can implement only one story at a time, until the story is finished or abandoned or until time runs out. We want to avoid having several stories "in progress" at the same time, to avoid the overhead of constantly switching from one task to the other. Because the only reliable measure of progress is finished stories, we prefer small stories to give us the most detailed feedback.

When the team is ready to start the implementation, the coach turns the hourglass upside down. One side of the hourglass is marked, so that it's not too difficult to remember which side is 'up'. The

hourglass is stopped (laid on its side) when the team declares the story implemented.

The stories are all simple tasks like finding a missing card in a deck, sorting a deck of cards, inflating balloons, folding a paper hat... You can find a detailed description of the different stories in the next section.

Acceptance test

When the team says the story is finished, the coach stops the time, and performs the acceptance test. The coach should pay attention to the quality of the implementation.

When the story is implemented to his satisfaction, the coach can add its business value to the score sheet.

If the story is not really finished in the coach's opinion, the team doesn't earn the business value. The players should have a look at the time. Maybe it's better to warn the customers, and ask *them* whether this story has to be finished, or if it's better to drop it. If the customer decides the story must be completed, the coach starts the hourglass again and lets the players complete the story.

Abandoning a story

If a story takes too much time, the coach may allow the team to stop its implementation. When that happens, the developers have to report the problem to the customer. This means that the whole team moves to the planning table, so that it's clear for everybody that the team has switched roles. The coach now explains the problem to the customers, who decide what to do.

They can choose to put aside the story, in which case the business value is not earned. Or they can choose to continue with the implementation until it's finished.

The team moves back to the developer table, and the implementation proceeds.

Tracking

Halfway through the iteration (the hourglass is half empty), the coach halts the implementation to let the team assess how they're doing. Half of the time has elapsed. How many stories did they implement? How many story points have been implemented?

Developers re-estimate points per iteration

We can estimate that by the end of the iteration; about twice that amount of effort will be spent. We now have a pretty good idea of which stories will be finished at the end of this iteration.

For example: initially you guessed for the team that they could perform 12 points worth of stories. About halfway they have implemented 5 points worth of stories. That indicates that they will probably implement about 10 points worth of stories by the end of the iteration. The customer should be told about this updated estimate.

Customer updates iteration plan

The team moves to the planning table, they are now in the customer role. The customer should modify the iteration plan. In this case, they will most likely have to drop at least 2 points worth of stories. The customer might modify the order of the stories (e.g. bring shorter stories, which are more likely to get finished, forward), drop some stories, and replace a story; anything that can improve the business value that can be earned in the estimated time.

After the customer has updated the iteration plan, the team moves back to the developer table, and they are back in the developer role. The team can continue with the implementation of the updated iteration plan.

Team reviews iteration

After the iteration has been implemented, the team quickly reviews the iteration. How did it go? What could they have done better? Where did they have difficulties? These and other questions are discussed with all the other teams at the iteration debriefing.

The team should now review the estimates of the stories they implemented. First, they lay out all the story cards by complexity, as

they did when first estimating complexity. Now, the coach should ask the question "do these estimates still look consistent?" Maybe some stories were grossly underestimated or overestimated. Use the same technique as when making the original estimates:

Lay out the implemented story cards according to experienced difficulty;

The simplest story that initially was assigned 2 points is not changed

If necessary, give other stories a new 1-6 estimate, relative to the 2-point story.

It's important that everybody's happy with these new estimates. In the following iterations the team will estimate new stories by comparing them with these stories.

Debriefing

To start the debriefing, it is best to take the participants away from the developer table. Let them take their chairs and sit in one circle or let them come forward to do a standup.

The score

Select the "Game Score page" in the presentation.

Each team now has to come up with a team name. Ask each team for their team name (column 1), initial estimated velocity (column 2), earned business value (column 3), and measured actual velocity (column 4).

Questions concerning Estimation

How did you estimate how long a task would take? For example: the build-a-house story.

How did you react when you noticed that 2 stories looked exactly the same to you? For example: the find-cards stories or the balloon stories.

Were the estimates the same for the different teams? For example the folding stories.

Real-life-customers: estimating is a typical developer task. How did you feel making the estimates?

Real-life-developers: do you feel the real-life process of estimating is easier or more difficult?

How did you feel about the accuracy of the separate estimates? Before / after implementation? Did you have to make large changes to the estimates after implementation?

Were there any surprises when you started implementing stories? What did you do?

Were there any teams that gave up a story during implementation?

Questions concerning Iteration Plan

How did you decide which was the most important story (=the first story of the iteration plan)?

Which strategy did you use to set up the iteration plan? For example: put the high value stories first, or put the stories with the highest value per effort ratio first, or put the lowest risk stories first…

Real-life-developers: How did you feel doing a typical customer task like creating an iteration plan and deciding which stories are the most important?

Real-life-customers: does this resemble the way you make a plan in real life?

Were there teams that changed the iteration plan during implementation?

How was your confidence in the iteration plan as a whole? Where you able to implement it? What went well? What went wrong? Where there any changes during the iteration plan?

Measure velocity and business value

The team will probably not have implemented the stories they originally planned. They may have implemented more or less. We want the team to plan more accurately in the next iteration.

At the beginning we had to guess how many points worth of stories we could implement in one iteration. But now we know how much we can implement in one iteration of 180 real seconds: the sum of the points of all implemented stories. **This is the team's velocity!**

VELOCITY = # of story points / iteration

We know how much business value we have delivered in this iteration.

The velocity and business value earned by the different teams is written on the table in the presentation.

Game iteration 2

Customer writes stories

The team members act as developers. They sit at the developer table.

The team takes the next pack of story cards from the bag. The pack for the second iteration contains 7 cards. All the stories that didn't get implemented in the previous iteration are available too.

Developers estimate stories

The team members act as developers. They sit at the developer table.

The team estimates the new story cards.

All the implemented stories are still laid out according to their points estimate.

To estimate the other stories, the team lays them out next to the implemented stories. Take each story to estimate and put it at the same height as one of the implemented stories that seems to require about the same complexity. If there's no good fit, put it between the two stories that come closest.

Once you have the stories laid out according to effort, it's easy to assign the estimates: if a story is next to an implemented story, give it the same number of points. If a story falls between two implemented stories, give it a value between the points value of the two implemented stories.

After you've assigned points to all stories, look at all the stories to see if the estimates look consistent. Make any changes you need.

When the team is done, they hand the estimated stories to the customer.

Customers choose and prioritise stories

The team members act as customers. They stand at the planning table.

The team chooses and orders the estimated stories. **The team may choose stories whose total points do not exceed the velocity of the team.**

The teams can try to adjust their strategy to select and prioritise the stories for implementation.

To emphasize the importance of estimating and planning correctly, **the scoring rules change**:

- If the team has time left, they can ask the customer for more stories. These stories are only worth half the business value written on the card. That's because "our salespeople had not expected these stories to be available. They haven't marketed them, so there will be fewer customers who will buy the product because of those new features."
- If the team runs out of time, half of the business value of the planned but not implemented stories is deducted from the score. That's because "our salespeople had promised these stories to customers. Not delivering reduces our income and reputation."

Developers implement stories

The team members act as developers. They sit at the developer table.

The team again tries to implement as many stories as possible in 180 seconds, following the same rules as in iteration 1.

Again, the coach stops the simulation when about half of the time has elapsed, so that the players can verify if they're on schedule. They should have implemented stories worth about half their velocity.

If they're behind, they should warn the customer. They can switch to the customer role and update the iteration plan.

If they're ahead, they can tell the customer and continue implementing stories. They will probably implement all scheduled stories before the end of the iteration. When that happens, they can switch to the customer role to select more stories to implement in this iteration. The new scoring rules mean that those extra stories are only worth half their indicated business value.

Measure velocity and business value

If everything goes well, the team has come closer to implementing the iteration as planned. The **new velocity** is computed by adding the estimates of the implemented stories.

Debriefing questions after second iteration

If you compare the first and the second iteration, are there any differences?

How accurate was the iteration plan this time? How far off were you?

Did your velocity go up (asked for more stories) or down (had to drop stories)?

Game iteration 3

Customer writes stories

The coach gives the team a third pack of story cards. The team uses the new stories and those that didn't get implemented in the previous iteration to plan an iteration.

Developers estimate stories

Again, estimate by comparing with the stories that have already been implemented. Try to keep the estimates consistent.

Customer plans an iteration

The customer chooses and prioritises stories. The total estimated points of the selected stories may not exceed the velocity of the previous iteration.

Developers implement stories

The team implements each planned story in turn, following the same rules as in the previous iteration.

Customer changes the plan (optional)

After the first story has been implemented, the coach halts the game. He introduces a new story card with very important last-minute feature requests. He asks the team of developers to estimate this new story. This shouldn't take much time because the team should be proficient at estimating by now.

Emphasize that the story's business value can only be earned if the team follows the process. We want the team to remain disciplined when faced with unexpected events.

The coach now asks the team to re-plan their iteration to take the new story into account. The team act as customers.

Before the new story can be added to the iteration plan, some other stories will have to be dropped; otherwise the team can't finish their work in time. For example, if the new story is estimated at 5 points, the

team will have to drop stories whose estimates are at least 5 points. Of course, we only exchange stories if the new stories bring us more business value than the dropped stories.

The whole re-planning session shouldn't take very long. After the plan has been updated, the team go back to being developers and continue implementing.

> WHY?
>
> This shows how a team can respond very quickly to changes in requirements, even in the middle of an iteration. We learn that we can't add more work to an iteration; we can only exchange stories for an equal amount of work. We only exchange stories if we can increase business value. Thus we keep the effort and release date constant, while increasing business value; and it doesn't cost us much effort.

Do this only when no unexpected things have happened during the first 2 iterations. The team has to be familiar with the new way of planning before they can start to think to try to break it.

Game end

The game host announces the winning team. The team with the most business value wins.

> WHY?
>
> Be very careful with what you measure, because people will act so as to maximize the parameter you measure. What we want to maximize is value created for the team, the company.
>
> In real life, stories often don't come with a convenient numerical value. Nevertheless, this variable is what we need to monitor to see if the team is (not) doing well. We don't suggest you perform complex value estimations. Just like with the cost estimates, an arbitrary but consistent value estimating method can tell us if the team is providing more or less value than in the previous iterations.

The use of velocity in the planning game should improve the estimates made by the team. At the end of the game they should notice that their iteration plans are becoming more accurate. They should be able to reach their targets.

Debriefing questions at the end of the game

- Compare the velocities of each team. What does the velocity tell you about the performance of the team?
- Compare the business value earned by each team. What does the business value tell you about the performance of the team?
- Did your estimates get more precise as the game went on?
- Did your velocity stabilize or did it fluctuate wildly?
- Could you see your team working with stories? Would you be able to write small stories (e.g. taking less than a few days to implement) that deliver some business value?
- Could you see your team estimating by comparing with other stories? Which estimation method do you use now?
- What were your impressions of the customer and developer roles? Are the responsibilities well defined? What do you think about the

way the two roles interact? How are these responsibilities divided in your team?

- Could you apply the planning game to your projects? Why? Why not?
- How far ahead do you plan currently? Could you divide your work in small iterations? If you need a longer plan, could you divide your iteration in several small iterations and then do the planning game for each of these iterations? Would that yield a good plan?

The Props

Do you have everything?

You can use the "Materials.xls" sheet to check if you have all the materials for the game.

The bag

Each team gets a closed when the game  begins.

The bag contains

- 3 sets of story cards: 10 cards for the first iteration, 8 for the second and 8 for the third.
- 3 score sheets, 1 for each iteration
- 1 Customer responsibilities sheet, to put on the planning table
- 1 Developer responsibilities sheet, to put on the developer table
- 6 pencils
- 1 marked hourglass
- 30 A5 folding papers (including blue and yellow paper)
- 1 deck of cards
- 3 dice

- 25 balloons
- 3 pieces of string: 1 of 40 cm, 1 of 60 cm, 1 of 80 cm
- 1 example hat, 1 example boat

The Score sheets

There's a Score Sheet for each iteration. When the iteration plan is ready, it is written down on the Score Sheet. For each story in the iteration plan, write down the story number and a short description of the story.

During the implementation, the numbers are filled in. When a story is implemented, write down its estimated time and its business value, and cumulate them in the 'total' column.

The Game Rules document

You can give each participant a "Game Rules" document at the start of the session, before you start with the presentation. It will be a guideline that they can use when they get lost.

Customer/Developer Responsibilities Sheets

In XP, the responsibilities for the customer and the developer are separated very strictly. It's very important that this fact is emphasized in the XP Game.

The team plays the role of both developer team and customer team, so that they can experience the 'other point of view'. But they should never forget which role they're in.

The Customer/Developer Responsibilities Sheets are visible on the planning/developer tables. When the team acts in a certain role, the coach has to make sure they are at the correct table and address them with the correct role name.

On each Customer/Developer Responsibility Sheet you can see what your responsibilities are. Sometimes, when you're a developer, you have to pass control to the customer (see page 16). In that case, the coach tells the team to look at the Customer/Developer Responsibility Sheet, and to move to the correct table, so that the team clearly sees that they switched roles.

Responsibilities of the Customer

Write and explain stories

Make Iteration Plan: choose and prioritise stories

Define acceptance tests

Perform acceptance tests

Decide what to do if problems are reported

Responsibilities of the Developer

Estimate stories

Implement stories

Track and report progress during implementation

Report problems during implementation

The Hourglass

How do we start/stop the time?

If you look at the picture, you can see that the blue hourglass is running. The green one is stopped.

The hourglasses are marked 'up', so that we can remember which side is 'up' if we have to resume it.

The Story Cards

There are 28 cards for each team. The cards are assigned to the **3 iterations**. The first iteration has 10 cards, the second 8, and the third

8. There is 1 card marked "Extra". This is the card that can be used in the third iteration to change the iteration plan during implementation phase.

The team will probably have some questions about the assignments on the cards. It's important that the coaches for the different teams use the same criteria.

We can divide the cards in several groups. Cards of each group have similar, but not necessarily equal assignments.

House of Cards

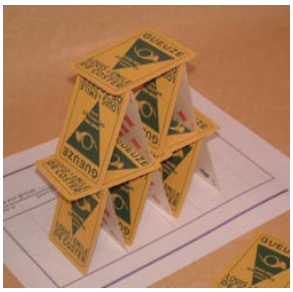
Stories:

Story 1: Build a 1-story house.

Story 3: Build a 2-story house.

Story 4: Build a 4-story house.

Estimation tips: This is how a 1-story house and a 2-story should look like:



Don't let the players spend too much time on spikes: the time is limited!

How easy or difficult it is to build a house of cards depends on the quality of the cards. Building a house with new cards is more difficult than building one with old cards. Building a 4-story house is almost impossible.

When the team decides to implement the 4-story house story, keep an eye on them. Try to avoid them getting stuck on trying to build the house of cards and not delivering any value. Keep telling them that it is possible to stop the implementation of the story and to ask the customers what they think they should do.

Start of Implementation: Put the deck of cards in the middle of the table. Start the hourglass.

Acceptance tests: When (and after) the hourglass is stopped, the house must be standing up.

Finding Cards

Stories:

Story 5: Find 1 missing card in a deck of cards, exactly 1 card is missing.

Story 6, 7, 8: Find 2 missing cards in a deck of cards, exactly 2 cards are missing.

Estimation tips: if you sort the deck, finding one or two missing cards takes about the same time.

Start of Implementation: First, take care that you shuffle the cards before the team starts with the implementation. Make sure the deck is complete. Pick one (or two) cards from the deck, and hide them.

Acceptance test: When the hourglass is stopped, nobody should touch the cards anymore. Then, the team must be able to say which card(s) the coach is holding. If it's not correct, they must proceed with the implementation. It's possible they want to give up (see page 16)

Sorting Cards

Stories:

Story 9: Sort 13 cards in sequence '1-10-J-Q-K. You'll only get these 13 cards.

Story 10, 11: Sort a full deck of cards.

Estimation tips: '1-10-J-Q-K' is how Belgian cards look, it means: 1 to 10 and then Knave, Queen and King.

Start of Implementation: First, take care to shuffle the cards before the team starts with the implementation. Make sure the deck is complete. Put the deck of cards in the middle of the table. Start the hourglass.

Acceptance test: When the hourglass is stopped, check if the cards are properly sorted. When the cards are not sorted correctly, you can't accept the story.

Balloons

Stories:

Story 12, 13: Inflate 5 balloons to a circumference of at least 40 cm and tie a knot in each

Story 14: Inflate 10 balloons to a circumference of at least 40 cm and tie a knot in each

Story 15: Inflate 5 balloons to a circumference of at least 60 cm and tie a knot in each

Story 16: Inflate 5 balloons to a circumference of at least 80 cm and tie a knot in each

Story 17: Burst 20 balloons

Estimation tips: There are measuring ropes in the bag: one of 40 cm, one of 60 cm, and one of 80 cm. If the players ask, you can show them exactly how big you expect the balloons to be. Most people are surprised that 40 cm really not very big. Until they start actually doing it... With the balloons I use, 80 cm is impossible.

Acceptance tests: Use the pieces of string to see if all the balloons are big enough. Be strict.

If there's a balloon that's not big enough, you can't accept the story. That means you have to start the time again, so that they can inflate the missing balloons. Or they can give up (see page 16).

Sums

Stories:

- Story 18: Make 20 additions. All numbers are between 0 and 100.

- Story 19: Make 20 subtractions. All numbers are between 0 and 100.
- Story 20: Make 40 additions. All numbers are between 0 and 100.
- Story 21: Make 40 additions. All numbers are between 0 and 1000.

Estimation tips: You have printed sum-cards, which contain the calculations they have to make. When they start implementing a sum-story, you give them the correct card, and they have to write down the answers on the card, so that you can check them easily. The team is allowed to tear up the card to divide the work among the players. They are not allowed to use any calculators.

Acceptance tests: For each story, you also have a card with the solution. This makes it possible to check if the answers are correct. If one of the sums on the card is not correct, you can't accept the story.

The sum stories always turn out to be more difficult than expected. Story 21 is almost impossible.

Folding

Stories:

Story 22: Fold 5 hats

Story 23: Fold 10 hats

Story 24: Fold 10 blue and yellow hats

Story 25: Fold 10 yellow hats

Story 36: Fold 5 blue boats

Estimation tips: The bag contains an example hat and an example boat. The team is allowed to look at them, and even to disassemble them. For Story 24, they have to fold 10 hats, with at least one blue and one yellow hat. You have provided them with blue and yellow folding paper.

Start of implementation: Most teams get very creative for this story. Usually, one or more players know how to do the folding, and others don't.

Acceptance tests: The resulting hats and boats should look exactly the same as the examples.

Three-dice score

Stories:

Story 27: Score at least 200 with 3 dice in one throw. 1 counts for 100, 6 for 60.

Story 28: Score at least 100 with 3 dice in one throw. 1 counts for 100, 6 for 60.

Estimation tips: These stories usually require some explanation. Throw 3 dice. The result is calculated like this:

1 thrown	→	100	points
2 thrown	→	2	points
3 thrown	→	3	points
4 thrown	→	4	points
5 thrown	→	5	points
6 thrown	→	60 points	

Some examples:

1, 2, 5 thrown	→	107	points
1, 6, 6 thrown	→	220	points
2, 2, 2 thrown	→	6 points	

Don't allow the team to do complex statistical analysis concerning the probability of throwing the required combination, that's considered to be **big up-front design**! We don't have time for that.

Start of implementation: They are allowed to work together, if they find a way to do that. But the 3 dice must be thrown in 1 throw.

Acceptance tests: If the players think they succeeded, they have to yell 'stop'. You stop the time and then you check if it's ok. If somebody messed up in the mean time, you can't accept the story.

Relations between the cards

Same Story Cards

You'll notice that the stories on some of the cards are exactly the same.

In the first iteration, there are several cards that contain exactly the same story. This allows the team to speed up the estimation round. It's also an important exercise in trying to estimate in a consistent way.

Maybe the team members will be surprised that for some of those equal stories the business value is different. It's important that they realize that the technical difficulty of a story is completely independent of the business value the story has for the customers. In real life, 2 stories can technically be implemented in exactly the same way (e.g. add a particular simple but persistent attribute to an existing object), but they can have a different value for the customer.

In the second iteration, there are several stories that are the same as stories they already had in the first iteration. Again, the stories have to be estimated consistently.

The following cards have the same stories:

Story 2, 3: Build a 2-story house (iteration 1 and 3)

Story 6, 7, 8: Find 2 missing cards (2 in iteration 1, 1 in iteration 3)

Story 10, 11: Sort a full deck of cards (iteration 2 and 3)

Story 12, 13: Inflate 5 balloons to 40 cm (both in iteration 1)

Story 23, 24, 25: Fold 10 hats (iteration 1 and 3)

Similar Story Cards

The cards that belong to the same group have similar assignments. Try to estimate those cards in a consistent way.

If a story is about twice as complex as another story, its estimate should be twice the estimate of the other one.

Impossible cards

Several cards are *almost* impossible to implement. It's possible that the team realizes this immediately, but it's also possible that they think they can implement it.

All the impossible cards have a very high business value.

The impossible stories are:

Story 4: Build a 4-story house (iteration 1)

Story 18: Inflate 5 balloons to 80 cm is impossible with the balloons I use (in iteration 2)

Story 21: 40 difficult sums (iteration 1)

Summary of the cards per iteration

Cards for iteration 1

4	house of cards	Build a 4-story house of cards	500
5	finding cards	Find 1 missing card in a deck of cards. Exactly 1 card is missing.	100
7	finding cards	Find 2 missing cards in a deck of cards. Exactly 2 cards are missing. The missing cards are "black" cards.	300
8	finding cards	Find 2 missing cards in a deck of cards. Exactly 2 cards are missing.	100
12	balloons	Inflate 5 balloons (to a circumference of at least 40 cm) and tie a knot in each	100
13	balloons	Inflate 5 balloons (to a circumference of at least 40 cm) and tie a knot in each	300
18	sums	Make 20 additions. All numbers are between 0 and 100.	100
21	sums	Make 40 additions. All numbers are between 0 and 1000.	300
22	folding	Fold 5 hats.	300
23	folding	Fold 10 hats.	300

Cards for iteration 2

1	house of cards	Build a single story house of cards	200
9	sorting cards	Sort 13 cards in sequence (1-10-J-Q-K). You'll only get these 13 cards.	300
10	sorting cards	Sort a full deck of cards in ascending order.	300
14	balloons	Inflate 10 balloons (to a circumference of at least 40 cm) and tie a knot in each	300
16	balloons	Inflate 5 balloons (to a circumference of at least 80 cm)	400

		and tie a knot in each	
19	sums	Make 20 subtractions. All numbers are between 0 and 100.	200
20	sums	Make 40 additions. All numbers are between 0 and 100.	300
26	folding	Fold 5 blue boats.	300

Cards for iteration 3

3	house of cards	Build a 2-story house of cards. Only "black" cards can be used.	300
6	finding cards	Find 2 missing cards in a deck of cards. Exactly 2 cards are missing.	300
11	sorting cards	Sort a full deck of cards. Colours also have to be sorted: ♦♥♣♠	400
15	balloons	Inflate 5 balloons (to a circumference of at least 60 cm) and tie a knot in each	400
24	folding	Fold 10 blue and yellow hats.	400
25	folding	Fold 10 yellow hats.	200
27	three-dice score	Score at least 200 with 3 dice (in one throw). 1 counts for 100, 6 counts for 60.	400
28	three-dice score	Score at least 100 with 3 dice (in one throw). 1 counts for 100, 6 counts for 60.	100

Extra card

17	balloons	Burst 20 balloons	500

Velocity

The XP planning game uses the developer's estimates of how difficult stories are to implement. When combined with the team's "velocity", these estimates allow the customers to define realistic and predictable iteration plans, which deliver the largest possible "business value". The team's velocity is measured after each iteration and should make subsequent iteration plans more predictable.

Iteration 1

The first iteration is the most difficult: you have no experience to draw upon, you don't know your velocity and you don't know anything about the difficulty of the stories.

The team can make a guess, but it'll never be more than a guess. Halfway through the iteration the team can assess how many points worth of stories they have implemented. At the end of the iteration they will probably implement about double that amount worth of stories. So, halfway through the first iteration you have a pretty good idea of how much work you can deliver.

At the end of the iteration you have some solid numbers: the estimates of the stories that were completed. Add these estimates and you have your **team velocity**. You now have a good idea of how much work you can perform in one iteration. Boy, are we glad the guessing is over!

VELOCITY = # of story points / iteration

Iteration n

For the following planning games, you have this solid indication of your team's ability to deliver effort: **the velocity**. This time round, your customers can choose stories whose estimate does not exceed your team's velocity. A typical team rapidly learns to estimate task complexity well, so the velocity should lead to feasible, predictable plans. And it's really easy to track. No complex formulae to memorize!

Can this simple scheme really work? It will only work if **developers estimate story complexity consistently**. Developers should estimate story complexity by comparing with implemented stories. If a story is about as complex as another story, give it the same estimate number. If the story is twice as difficult as another story, give it twice as many points. Story estimates don't have any unit: they're just indications of relative implementation effort. We call the unit **story points**.

How does velocity work?

After each iteration, the velocity gets updated. You can just use the previous iteration's velocity. Or you could get fancy and use a weighted average of the previous few iterations' velocity to even out the spikes.

If your team is overly optimistic when estimating; if you run into unexpected problems; if someone leaves the team; if someone new joins the team; if the structure of your code deteriorates; if your developers get distracted by other tasks... you might not be able to complete all planned work. You will have to tell your customer you can't implement all planned stories. Your team velocity will go down. You will get fewer tasks in the next iteration. You get some time to handle whatever is slowing you down.

If your team is pessimistic; if the structure of your code improves; if you find more opportunities for reuse; when new team members get up to speed... you will finish your work sooner than planned. Your team will ask more stories from the customer and your velocity goes up. You get more tasks for the next iteration, but you can handle it.

All these diverse factors influencing your development, and one easily computed number sums it all up! As your velocity stabilizes, the planning game can deliver more accurate iteration plans. That's the beauty of it: **if your estimates are consistent, your plans will be relatively correct.** And making consistent estimates is easier than making correct estimates.

Velocity calculations

You don't have to be a math wizard to calculate your velocity. A few additions will do. You can perform some further calculations upon your velocity.

We don't track velocity individually. Velocity is how much work the team can do in an iteration. Because we aggregate the numbers for a whole team and a whole iteration, estimation errors will cancel out.

If a team member leaves (permanently or on holiday) you can lower team velocity. If the developer will be gone half of the iteration, subtract half of their part of the team velocity. Nothing too complicated to calculate, just enough to warn your customer you may get less done this iteration. For example one of the five developers in the team will be on holiday for half the iteration. Subtract 1/10th from the velocity.

If a new developer joins the team, by how much do you increase team velocity? The rule with velocity is: **don't guess; don't calculate; measure**! Leave the velocity as it is. After the iteration you can measure this developer's contribution and incorporate it into the team velocity. The impact of a new team member will not be large at first (it may even be negative as other developers help the new team member to learn) so you should expect a dip, followed by a slow rise in team velocity.

References

[Beck 2000] "Extreme Programming Explained", Kent Beck – Addison Wesley 2000

[Beck 2001] "Planning Extreme Programming", Kent Beck & Martin Fowler – Addison Wesley 2001

[Kerth 2001] "Project Retrospectives", Norman L. Kerth – Dorset House 2001

About the authors

Vera Peeters is an independent consultant. She runs her own company TRYX. She has more than 15 years experience in developing software systems, especially object-oriented development in all kinds of high-technological environments. She has been practicing agile ways of working since 1999. Vera has presented workshops at several conferences: XP200X, XPUniverse, OT200X, XPDays (Benelux, London, Germany).

Vera can be contacted at vp@tryx.com or http://www.tryx.com/

Pascal Van Cauwenberghe is an independent consultant who has more than 15 years of experience in delivering systems as a programmer, designer and manager. He has applied agile methods since early 2000. He currently helps teams to improve their project management and development process skills.

Pascal can be contacted at pascal@nayima.be or http://www.nayima.be/

Vera and Pascal are co-organizers of the XP Days Benelux. In 2001, they founded the Belgian XP User Group.

If you have any remarks about the game, feedback after playing, tips for improvement, questions or bugs, contact the authors.

Have fun!

Vera and Pascal

License

Vera Peeters
vera.peeters@tryx.com
Tryx
http://www.tryx.com

Pascal Van Cauwenberghe
pascal@nayima.be
NAYIMA
http://www.nayima.be