XP Game

The Manual

Authors: Vera Peeters and Pascal Van Cauwenberghe

© Copyright Tryx bvba and Nayima bvba.

Permission to copy, use and distribute this document is granted provided this copyright notice appears in all copies.

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

We hope you enjoy playing the game!

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 a release 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 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 "release" 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.

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

Game requirements

Players

- Participants should know what "planning game", "velocity", "customer" and "developer" mean in an XP project. See, for example, [Beck 2000] and [Beck 2001].
- 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.
- Each team requires a coach. If there are more teams than coaches, some coaches may have to assist more than one 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.
- The coach answers questions about the game. The coach emphasizes which role the team is currently playing. He can use the Customer/Developer Responsibilities Sheet 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.

Terminology

- **Story**: a short description of a feature that, when implemented, will provide some value to the customer.
- **Acceptance Test**: a test performed by the customer, to verify if the story has been implemented correctly.
- **Business Value**: the "value" of the implementation of a story to the customer.
- **Story points**: measures how difficult it is to implement a story.
- **Real time seconds**: the stuff you can measure with a stopwatch. We use an hourglass to measure, so that players aren't able to see how long each individual task takes to implement. We will use the velocity to translate between real time and story points for a whole iteration.

Duration

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

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 and celebrate the winning team
25	Final debriefing and discussion
180	

The game begins

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

Questions you should ask before the start of the game:

- Who's a developer in real life? Who's a customer? 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 coach and answer questions about XP.

Each team has a bag with game props. Empty the bag on the table. The team can have a look at the props. Pick the pack with story cards that is marked "iteration 1".

Game iteration 1

Customer writes stories

Team members act as developers. The coach turns the "Developer" side of the Customer/Developer Responsibilities Sheet up.

The bag holds 3 sets of story cards. The team can open the set for the first iteration.

The first set contains 12 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.
- 10-20-30-40-50-60-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.

The team estimates all of the story cards. The coach should remind the players not to take too much time in estimating or designing a solution.

One of the numbers 10-60 must be circled to indicate: “this story will take 10-60 seconds to implement”.

The team may also declare “this story is too big to implement in 60 seconds”. 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 can't be done in less than 60 seconds. 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 to keep your stories short…

The team can 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.

The coach should suggest to the team to estimate by comparing stories. If two stories seem about as difficult, use the same estimate. 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. So, 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.

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

Customers choose and prioritize stories

Team members act as customers. The coach turns around the Customer/Developer Responsibilities Sheet, so that the "Customer" side is now up.

The team selects stories with a total estimated time of no more than 180 seconds (the fixed length of an iteration). The coach writes that down on the Scoring Sheet.

The team orders the stories for implementation, taking into account the business value and the difficulty estimate. They try to make a planning whereby they have the best odds of earning a large business value. Each team will devise its own strategies. E.g. put the high value stories first, put the stories with the highest value per effort ratio first, putting the lowest risk stories first…

The result is a release plan, which is written on the score sheet. For each story, the coach writes down the story number, story description and estimate. During the implementation the implementation of stories is tracked on the score sheet. 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 us to intervene as soon as problems become visible.

Developers implement stories

Team members act as developers. The coach turns around the Customer/Developer Responsibilities Sheet, so that the "Developer" side is now up.

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

Executing the release 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.

If all stories have been implemented and the hourglass is not yet empty, the team, acting as customers, may select another story to implement. Don't forget to turn around the Customer/Developer Responsibilities Sheet, because the team is now re-planning, a customer responsibility.

Implementation of a story

For each story, the team can briefly discuss the story implementation. Team members sign up for the task. Many tasks can be implemented by the entire team.

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 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, blowing up balloons, throwing a number with dice… 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.

Tracking

Halfway through the iteration, the team can assess how they're doing relative to the plan. Half of the time has elapsed. Have they performed more or less than half of the stories? This will give the team a clear indication of their performance. If things don't go as planned they will be able to warn their customers.

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. The coach must turn around the Customer/Developer Responsibilities Sheet, 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. The business value is not earned. Or they can choose to continue with the implementation until it's finished.

The coach turns around the Customer/Developer Responsibilities Sheet again, and the implementation proceeds.

Debriefing

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, the balloons stories, or the roll-a-monkey 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?
- Were there any surprises when you started implementing stories? What did you do?
- Were there any teams which gave up a story during implementation? Were there teams which changed the release plan during implementation?

Questions concerning Release Plan

- How did you decide which was the most important story (=the first story of the release plan)?
- Which strategy did you use to set up the release 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…
- Developers: How did you feel doing a typical customer task like creating a release plan and deciding which stories are the most important?
- How was your confidence in the release plan as a whole? Were you able to implement it? What went well? What went wrong? Were there any changes during the release plan?

Introducing concept of Velocity

- How much did we do in this iteration? How many business points? How many estimated seconds?
- Which teams exactly implemented the release plan? Which teams did less? More?
- When we estimated, we tried to estimate in seconds, because that was the only thing we knew. Now, we make a difference between "estimated seconds" and "real seconds".
- Concept of velocity: How many estimated seconds can we implement during 1 hourglass?
- Velocity of the different teams: are they different? Why is that? Do the different teams have different estimates for the stories? How do the different release-plans look like?

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.

We now know how much we can implement in one iteration of 180 real seconds: the sum of the estimates 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 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

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

The team estimates the new story cards.

The coach should tell the team to estimate stories by comparing them with the stories they have implemented. E.g. if a new story is about as difficult as a 30 point story, estimate it at 30. This is the most difficult thing to explain: we're no longer talking about real time in seconds, we're just looking at relative complexity. The story cards don't mention seconds, only the numbers are printed on the cards. The coach should make it clear we're now talking about "story points".

Customers choose and prioritize stories

The team members act as customers

The team chooses and orders the estimated stories. **The team may choose stories whose total estimated time does not exceed the velocity of the team** (= the sum of the estimates of the implemented stories).

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

Developers implement stories

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

Measure velocity and business value

If everything goes well, the team has come closer to implementing the release 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 release plan this time? Did your velocity go up or down?

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 releases, to plan a release.

Developers estimate stories

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

Customer plans a release

The customer chooses and prioritizes 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 two new story cards with very important last-minute feature requests. He asks the team of developers to estimate these two new stories. This shouldn't take much time, the team should be proficient at estimating by now.

The coach now asks the team to replan their release to take the new stories into account. The team act as customers.

Before the new stories can be added to the release plan, some other stories will have to be dropped, otherwise the team can't finish their work in time.For example, if the two stories are estimated at 20 and 30 points, we will have to drop stories whose estimates are at least 50 points. Of course, we only exchange stories if the new stories bring us more business value than the dropped stories.

The whole replanning 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 repond very quickly to changes in requirements, even in the middle of a release. We learn that we can't add more work to a release, 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.

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 quantity you measure. What we want to maximize is value created for the team, the company.

In real life, stories don't come with a convenient numerical expression of their 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 release 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?

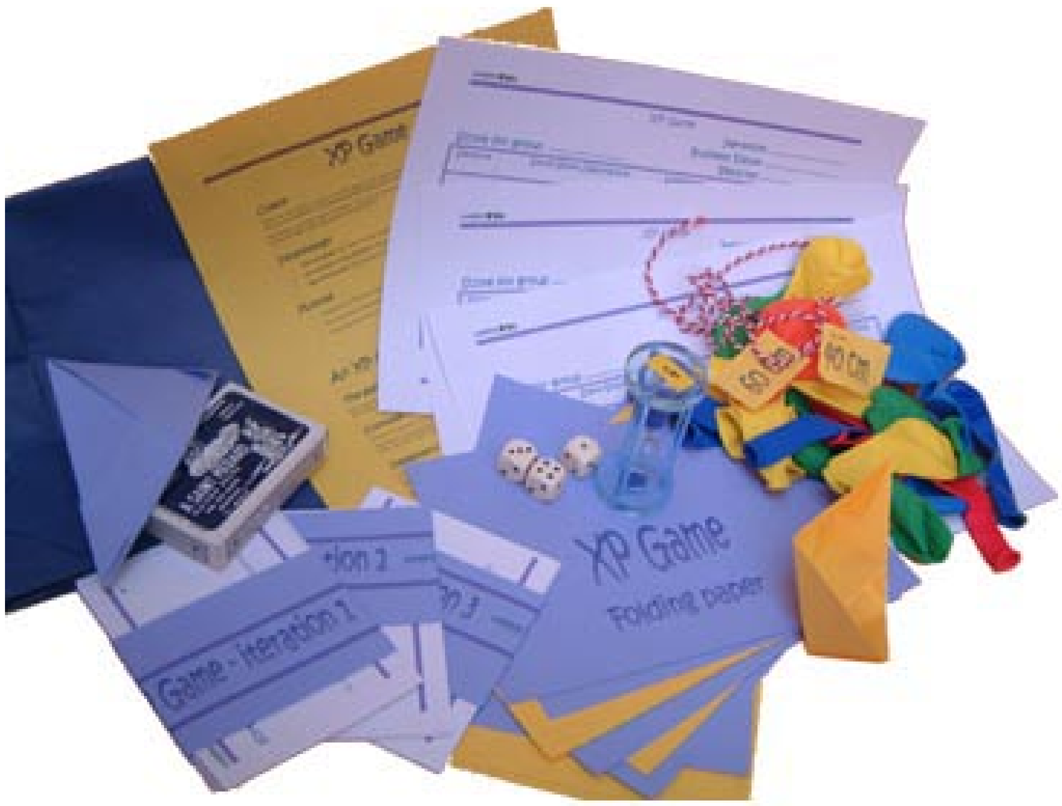
- Could you apply the planning game to your projects? Why? Why not?
- How far ahead do you plan currently? Could you divide your work in small releases? If you need a longer plan, could you divide your release in several small releases and then do the planning game for each of these releases? Would that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

The Props

The bag

Each team gets a bag at the start of the game.

The bag contains



- 3 sets of story cards: 12 cards for the first iteration, 7 for the second and 7 for the third. There are also 4 extra cards, which can be used when all available stories have been implemented and there is still time left in an iteration or to use as last-minute feature requests in iteration 3.

- 3 score sheets
- 1 summary for the coach
- 1 Customer/Developer responsibilities sheet
- 1 pencil
- 1 hourglass
- 20 folding papers
- 1 deck of cards
- 3 dice
- 15 balloons
- 1 piece of string of 40 cm, 1 piece of string of 60 cm
- 1 example hat, 1 example boat

The Score sheets

There's a Score Sheet for each iteration. When the release-plan is ready, it is written down on the Score Sheet. For each story in the release-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 Coach Summary

Make sure each coach has some time to look at the summary card before the game starts. Hold a short briefing session with the coaches just before the game.

Customer/Developer Responsibilities Sheet

In XP, the responsibilities for the customer and the developer are separated in a very strict way. 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 Sheet has 2 sides: a Customer Responsibility Side and a Developer Responsibility Side. When the team acts in a certain role, the coach should address them with the correct name, and he should also take care that the correct side of the Customer/Developer Responsibility Sheet is up.

On each side of the 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 11). In that case, the coach should turn around the Customer/Developer Responsibility Sheet, so that the team sees clearly that they switched roles.

Responsibilities of the Customer

- Write and explain stories
- Make Release Plan: Choose and Prioritize 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 30 cards for each team. The cards are assigned to the 3 **iterations**. The first iteration has 12 cards, the second 7, the third 7. There are 4 cards marked "Extra". If your team has implemented all the stories, and there's still time left, you can give them an 'extra' card to estimate and implement. This rarely happens.

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 2, 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: estimating time is limited!

How easy or difficult it is to build a house of cards normally 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.

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, 6, 7: Find 1 missing card in a deck of cards, exactly 1 card is missing.
- Story 8, 9, 10: 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 special care that you mix the cards before the team starts with the implementation. It's also best if you check if 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 11)

Sorting Cards

Stories:

- Story 11: Sort 13 cards in sequence '1-10-J-Q-K. You'll only get these 13 cards.
- Story 12: Sort 1 sequence of 13 cards in a specific colour. You'll get a full deck of cards.
- Story 13, 14: 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 special care that you mix the cards before the team starts with the implementation. It's best if you check if 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, you check if the cards are properly sorted.

Balloons

Stories:

- Story 15, 16: Inflate 5 balloons to a circumference of at least 40 cm and tie a knot in each
- Story 17: Inflate 10 balloons to a circumference of at least 40 cm and tie a knot in each
- Story 18: Inflate 5 balloons to a circumference of at least 60 cm and tie a knot in each

Estimation tips: There is a rope of 40 cm and one of 60 cm. If they 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, 60 cm is almost 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.

Three-dice score

Stories:

- Story 19: Score at least 100 with 3 dice in one throw. 1 counts for 100, 6 for 60.
- Story 20: Score at least 180 with 3 dice in one throw. 1 counts for 100, 6 for 60.
- Story 21: Score at least 300 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 complicated 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.

Monkey dice

Stories:

- Story 22, 23: Roll a monkey 5 times using 1 dice.
- Story 24, 25: Roll a monkey 5 times using 2 dice.
- Story 26: Roll a monkey 10 times using 2 dice.
- A "monkey" means throwing a given number with one or two dice. For example: throwing two ones, also known as "snake eyes".

Estimation tips: The monkeys differ for the different stories. Each story says explicitly which number acts as a monkey. Using 1/2 dice means 1/2 dice must be thrown at a time.

Start of implementation: For this story, it's not possible to do the acceptance test afterwards. The coach counts the number of valid throws during the implementation. Therefore, it's very important that the throwing is done in a disciplined way. Players and coach can agree that the coach gives a signal after checking a throw, before the players throw again.

Acceptance tests:

Folding

Stories:

- Story 27: Fold 5 hats
- Story 28, 29: Fold 10 hats
- Story 30: Fold 5 boats

Estimation tips: The bag should contain an example hat and an example boat. The team is allowed to look at them, and even to disassemble them.

Start of implementation: Most teams get very creative for this story. Usually, one or more persons 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.

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 meaning 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. If the team implemented one of those stories in the first iteration, it's possible that they now have a better understanding of the difficulty of that story. In that case, they are allowed to update the estimate, so that it better reflects the complexity of the story.

The following cards have the same stories:

- Story 2, 3: Build a 2-story house (iteration 1 and 3)
- Story 5, 6, 7: Find 1 missing card (2 in iteration 1, 1 extra)
- Story 8, 9, 10: Find 2 missing cards (2 in iteration 1, 1 in iteration 3)
- Story 13, 14: Sort a full deck of cards (iteration 2 and 3)
- Story 15, 16: inflate 5 balloons to 40 cm (both in iteration 1)
- Story 22, 23: Roll 5 monkeys using 1 dice (iteration 2 and 3)
- Story 24, 25: Roll 5 monkeys using 2 dice (both in iteration 1)
- Story 28, 29: Fold 10 hats (iteration 1 and 2)

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 60 cm is impossible with the balloons I use (in iteration 3)
- Story 21: Score 300 with 3 dice in 1 throw (iteration 2)

Summary of the cards per iteration

Cards for iteration 1

2 house of cards	Build a 2-story house of cards	300
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
6 finding cards	Find 1 missing card in a deck of cards. Exactly 1 card is missing.	200
9 finding cards	Find 2 missing cards in a deck of cards. Exactly 2 cards are missing.	300
10 finding cards	Find 2 missing cards in a deck of cards. Exactly 2 cards are missing.	100
15 balloons	inflate 5 balloons (to a circumference of at least 40 cm) and tie a knot in each	100
16 balloons	inflate 5 balloons (to a circumference of at least 40 cm) and tie a knot in each	300
24 monkey dice	Roll a monkey 5 times using 2 dice. A monkey is a 4.	300
25 monkey dice	Roll a monkey 5 times using 2 dice. A monkey is a 6.	200
28 folding	Fold 10 hats.	300
30 folding	Fold 5 boats.	300

Cards for iteration 2

1 house of cards	Build a single story house of cards	200
11 sorting cards	Sort 13 cards in sequence (1-10-J-Q-K). You'll only get these 13 cards.	100
13 sorting cards	Sort a full deck of cards.	300
17 balloons	inflate 10 balloons (to a circumference of at least 40 cm) and tie a knot in each	300
21 three-dice score	Score at least 300 with 3 dice (in one throw). 1 counts for 100, 6 counts for 60.	400
23 monkey dice	Roll a monkey 5 times using 1 dice. A monkey is a 5.	200
29 folding	Fold 10 hats.	200

Cards for iteration 3

3 house of cards	Build a 2-story house of cards	300
8 finding cards	Find 2 missing cards in a deck of cards. Exactly 2 cards are missing.	200
14 sorting cards	Sort a full deck of cards.	400
18 balloons	inflate 5 balloons (to a circumference of at least 60 cm) and tie a knot in each	400
19 three-dice score	Score at least 100 with 3 dice (in one throw). 1 counts for 100, 6 counts for 60.	100
22 monkey dice	Roll a monkey 5 times using 1 dice. A monkey is a 1.	100
26 monkey dice	Roll a monkey 10 times using 2 dice. A monkey is a 2.	300

Extra cards

7 finding cards	Find 1 missing card in a deck of cards. Exactly 1 card is missing.	300
12 sorting cards	Sort 1 sequence of 13 cards (1-10-J-Q-K ♥). You'll get a full deck of cards.	300
20 three-dice score	Score at least 180 with 3 dice (in one throw). 1 counts for 100, 6 counts for 60.	200
27 folding	Fold 5 hats.	200

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 release plans, which deliver the largest possible "business value". The team's velocity is measured after each iteration and should make each iteration 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.

In the first iteration, developers estimate the difficulty of the stories in "**perfect engineering days**". These are those rare days when you can work without being interrupted, without any other tasks to do, without any distractions. Developers divide the stories into small tasks that have to be done to implement the story, a sort of high-level design. Each task is then estimated in perfect engineering hours. The sum of the estimates of the tasks is then the estimate of the story.

To plan the release, you must know the ratio between real days and perfect engineering days. As a first approximation we usually take 3 real days per perfect engineering day. An average software engineer spends about 1/3 of his time on "useful work". In the simulation we only measure the actual implementation time. Thus, the players can estimate how long the stories will take, without having to take into account productive vs non-productive time.

How many stories can you implement in a given time? Say you perform an iteration of 20 working days with 6 developers. As a first guess, you can perform 20x6/3 or 40 perfect engineering days this iteration.

The customers selects 40 perfect engineering days' worth of stories for the team to implement. Keep in mind that this is not a plan you can count on, what with that *magic* number three in that formula. So, expect your plan to be wrong. You will implement more or fewer stories than planned this iteration. Concentrate on **completing** stories, rather than starting them all at the same time. If time runs out, at least you can deliver some completed stories. Incomplete stories are of no use to your customer.

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! Now we're working with real numbers you can forget all that stuff about perfect engineering days. From now on, we will measure story difficulty in "**story points**".

VELOCITY = # of story points / iteration

Iteration n

For the following planning games, you have this solid indication of your team's ability to deliver functionality: **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 formulas to memorize!

Can this simple scheme really work? Well, it will only work if **developers estimate story complexity consistently**. Developers should estimate story complexity by comparing with previously estimated 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 difficulty. We just call 'em points. Call them anything you like, just don't use the word "day"; it confuses customers.

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 month's 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 (but you will have implemented the most important 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 will get more work to do 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 release 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.

You could keep individual velocity numbers for each developer. This is the number of story points the developer can implement per iteration. The sum of the developer velocities is the team velocity. This can help you to allocate stories fairly at the start of the iteration: each developer may sign up for stories until his personal velocity is reached. Take care not to use only this number to evaluate this developer's productivity. On real projects the authors have never needed to track individual velocity numbers, tracking the team velocity is enough. Let the developers distribute the work fairly amongst themselves.

If a team member leaves (permanently or on holiday) you can subtract that developer's velocity from the team velocity. If the developer will be gone half of the iteration, subtract half of his 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 guy to learn) so you should expect a slow rise in team velocity.

Typical velocity questions

The players will ask the coaches about velocity. Here are a few typical questions and remarks we have encountered. Try to prepare a good answer to them.

- I've got this story estimated at 20, but it really took 50 seconds. Now I've got this comparable story. I **know** it will take 50 seconds, so I'll estimate it at 50.
- This story took 5 perfect engineering days. I know how long a perfect engineering day really takes, because we track developer activity. So, perfect engineering time corresponds to the real time the developer spent on implementing this story.
- I can do this story in 6 days. This other story took 6 days. It was estimated at two points. So I'll estimate this story at two points.
- I've got two stories (A and B) that were both estimated at two points. It turned out that the first story took 2 days to implement, and the other took 6 days to implement. So, those estimates are useless, because we now know they weren't of the same complexity. That means the velocity will be wrong. (If you have a good answer to this one, let us know!)
- I estimated story A at 2 points. I've now got stories B, C and D that are about the same. But now I know how to do them, it'll be easier. So I will estimate them at 1 point.
- Last iteration we completed 14 points instead of the 21 planned. But we had lots of unexpected problems, and X was off sick, and we had to do some other stuff. That won't happen again this month, so we can still plan to do 21 points.
- We have a fifth developer, so our velocity goes up by one fifth.

Appendix

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 who has been building systems with C++ and Java for more than 10 years. She has applied agile methods since early 2000. She currently coaches development teams to help them to improve their product and process.

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

Pascal Van Cauwenberghe is an independent consultant who has more than 10 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 pvc@nayima.be or http://www.nayima.be/

Or you can meet them at the Belgian and Dutch XP users groups (http://www.xp.be/ and http://www.xp-nl.org/).

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