# Managing the Bootstrap Story in an XP Project

**Jennitta Andrea**
ClearStream Consulting Inc.
1200 – 250 – 6th Avenue S.W.
Calgary, Alberta
Canada T2P 3H7
+1 403 264 5840
jennitta@clrstream.com

**ABSTRACT**
An ideal XP project is composed of stories defined by the customer that are of the right size and focus to plan and manage according to XP principles and practices. The reality of many XP projects is that the key story in the first release, the bootstrap story, is much larger than the rest of the stories. The bootstrap story represents the smallest deliverable kernel of the system that subsequent stories build upon incrementally. A large story creates a variety of problems: it does not fit into an iteration; there are a large number of tasks that are difficult to coordinate; and/or is too large to test adequately at the story/functional level. Teams new to XP find managing the bootstrap story especially challenging because they lack the experience required to deal with the additional planning complexity. A number of strategies exist to mitigate the problems caused by a large story. This experience report explains how our team 'cheated' in order to successfully manage a bootstrap story.

**Keywords**
Chapter, Bootstrap Story, Iteration Planning, Story, Story Granularity, Story Refactoring

## 1 INTRODUCTION

We have found a common pattern amongst a number of our XP projects. The first release of a system requires at minimum a thin slice of the core business process that the system supports. The customer describes this thin slice of behavior as a single story because they cannot conceive of any smaller unit of useful functionality to be delivered. We call this the *bootstrap story* because the end result is a minimal functioning system. This bootstrap story feels much larger and covers a wider breadth of the system than other stories, which incrementally add focused features to this established core. For example, the core business process supported by a billing system is to apply a costing formula to usage data based on business rules, and generate an invoice for each customer. The bootstrap story for the billing system describes the full business process within the simplest possible parameters: assume error free data, use the most basic costing formula, and generate a simple invoice with a single fixed format. Other stories incrementally build on this core to introduce more depth and breadth to the system: specific data validation, different costing formulas, processing corrections to previous invoices, and customizing the invoice formatting.

The focus of this paper is to share our experiences with managing the bootstrap story within XP projects. The problems in managing the bootstrap story in the first release of an XP project are described, followed by an overview of several techniques for mitigating these problems. A more detailed example is provided based on our current work on a billing system.

## 2 PROBLEMS

To understand the problems that are generally experienced with a bootstrap story, a quick review of the XP concept of a story is helpful. The term *story* describes an individual feature that represents real business value to the customer. The customer is responsible for defining the stories for their system. From a planning perspective, the story is the unit of prioritization, scheduling, and progress tracking that is visible to the customer. An XP project has frequent small releases, each of which contains a number of time boxed iterations. Iteration planning involves scheduling one or more stories in a particular iteration, based on the priority and size of the story. The entire story must be finished within the iteration it is scheduled for.

The root of all the problems created by the bootstrap story is that it is too large. The problems arise in three areas:

### Iteration Planning

The first problem that a bootstrap story creates for an XP team is in iteration planning. Although the bootstrap story is expressed in the simplest possible terms, implementing the thin slice of the process generally touches the system from end to end. While the effort to implement the bootstrap story may fit within the time box for the release, it generally does not fit into the time box for a single iteration. This makes iteration planning of the bootstrap

story more challenging than for other stories. Typically the team must treat the bootstrap story as a special case. From the perspective of a team new to XP, this is particularly vexing because the techniques they use on the first iteration don't necessarily help them on subsequent iterations. Or, put another way, the first iteration is much harder to plan and the team has few if any of the skills and/or experience necessary for dealing with the additional planning complexity. Double-Whammy!

**Task Coordination**

Task coordination is the second area that problems can arise. The bootstrap story is a large story that generates a large number of tasks. The tasks cover a relatively broad set of the core business processes, and require most of the depth of the architecture and technical infrastructure. We have found that our XP projects typically do not require micro-management of tasks to the extent that detailed grouping and dependencies of the tasks have to be worked out. For the bootstrap story, our typical approach of team members anarchistically selecting tasks defined for the current story, has resulted in integration difficulties because tasks are not well coordinated. It is not enough to ensure that everyone selects tasks from the same story because there is only one story. Extra overhead must be incurred to orchestrate the sequencing of cohesive tasks to ensure that the team makes progress towards a common sub-goal at any one point in time within the iteration.

**Story Testing**

The third problem experienced is that the granularity of the story testing is too large. The focus of a story/functional test is to validate the final, business-level output that the process generates from the input and the environmental context. The customer is responsible for specifying and signing off on functional tests. The bootstrap story is composed of many internal processing steps, which are not functionally tested in detail at the story level. While it is true that unit testing will ensure that individual internal processing steps function correctly, the customer is typically not involved at that level of testing.

## 3 PROBLEM MITIGATION

There isn't a 'one size fits all' solution for a project. Each project operates in its own context: team size, experience level, competitive pressures, development tools, etc. The context influences how the project is affected by particular bootstrap story problems. Each project needs to be assessed in terms of which of the problems exist, and which ones have the biggest impact. This section describes a number of techniques can be used to mitigate the bootstrap story problems.

**Simplify**

*Do the simplest thing possible.* In defining the scope for the bootstrap story, reduce the necessary core down to absolute bare bones. Ask questions like:

- Are there any steps in the process that can be done manually for the first release? If so, then remove the manual steps from the story. Make automating each of the manual steps a new story.
- Are any of the steps in the process optional? If so, then remove the optional steps from the story. Make each of the optional steps a new story.
- Does this story describe a single situation? If not, then focus the story on a single goal or situation. Make each of the other goals or situations a new story.
- Are we dealing with the simplest possible data formats? If not, define the simplest possible data formats for the story. If possible, assume that the data is error free, so that the story does not deal with data validation and/or cleanup. Define other stories that deal with the variations in data format and data validation.

**Refactor**

*Divide and conquer.* As opposed to simplification, which attempts to make a large story skinnier, refactoring endeavors to split a single large story into smaller individual stories. This step is especially challenging because it often forces the customer to reevaluate their definition of business value. Good facilitation skills, an open mind, and a bit of creative thinking are key to successful story refactoring, especially for the bootstrap story. For each candidate new story, ask:

- Does this story deliver true business value to the customer? If the customer answers 'yes', it is a legitimate story.

Keep in mind that for the bootstrap story, the answer to the first question is often 'no'. If so, then keep probing:

- What is missing from the story in order for it to deliver true business value?

Progressively add pieces back onto the candidate story, asking the above questions again. The end result will either be the original bootstrap story, which has been proven to be unfactorable, or a set of smaller stories, each of which provide business value.

**Cheat**

*Necessity is the mother of invention*. If the simplest possible bootstrap story still exceeds the time box for the iteration, the development team may be left with no choice but to treat the story as a special case – in other words, cheat! Cheating really amounts to innovating to overcome barriers to project success. Cheating can take many forms:

- Adjust the size of a single iteration so that the bootstrap story will fit into it (ie. the guideline that iterations should be roughly the same size is the barrier).
- Allow the bootstrap story to span multiple iterations

(ie. the guideline that stories should be completed within an iteration is the barrier).

- Refactor the bootstrap story into pieces that do not necessarily have business value to the customer (ie. the guideline that the customer refactors stories and ensures that each one has business value is the barrier).
- Expand the customers responsible to include the key unit tests for the sub processes contained in the bootstrap story (ie. the guideline that the customer is responsible for story/functional tests is the barrier).

## 4 EXAMPLE

Our team for the billing system project was composed of two full time customers and three senior Object Oriented (OO) developers. The customers were all new to both XP and OO. One of the developers was lead on several previous XP projects, so was designated as our XP coach. The other developers were familiar with XP concepts but had not practiced them before. This project possessed many of the necessary characteristics to succeed using an XP approach: short initial deadline, small initial scope, vague requirements (both initial and future), a trusting customer willing to try new things, and a development team experienced in the domain, the development tools, and with each other.

The core process for the billing system is to receive usage data in several formats and generate customer invoices based on the data and a variety of business rules (costing strategies, allocation strategies, etc). The first release of the system must build the kernel of the billing system - the simplest thin slice of the core functionality. The first story defined for the system was:

*Story 1: Calculate Benchmark Charge and Save In Standard Format*
*Load raw data from input files into the data warehouse. The data is assumed to be error free, so no data validation is required. Roll up the raw data into hourly intervals, and multiply each interval by the associated hourly price. These hourly charges are summed for the billing period to create a single charge per customer. The hourly prices are assumed to be available and error free. Output the generated charge into a simple file format. One file is generated per customer.*

This was the simplest possible definition of the core of the system. It consisted of the simplest form of input data, the simplest fixed costing strategy, and generated a simple form of invoice. The story is significantly bigger than the other stories that the customer defined.

Because this was the first XP project for most of the team, we struggled with this first story. We were more familiar with describing system requirements as use cases, and were trying to get our footing with the concept of what made a story different from a use case. Story1 felt too much like a use case, in that it covered the steps along the main path of the business process, rather than being a singularly focused feature. The team attempted to decompose the story into smaller stories, but continued to return to the fact that the customer did not see any business value in anything less. Once we were convinced that this was a legitimate story, we rapidly discovered the problems it created for us. The first problem we faced was iteration planning, so we innovated (cheated) in order to finish our plan.

One way that the development team cheated was to internally refactor the bootstrap story. Since the development team did the refactoring, we did not call the new entities stories, but rather called them *chapters*. The term chapter made it clear that it was a portion of a story rather than a legitimate story defined by the customer. We revised our processes for the first release:

- The large bootstrap story was assigned to the entire release instead of a single iteration, and its progress was tracked with the customer throughout the entire release.
- Each chapter was assigned to a separate iteration to provide the development team finer grained control over progress tracking and task allocation.
- Functional tests were created for the overall story as well as for each chapter. Both of these types of functional tests were the responsibility of the customer

The development team broke the bootstrap story into three chapters:

*Chapter 1: Load Data into Data Warehouse*
*Load raw data from input files into the data warehouse. The data is assumed to be error free, so no data validation is required.*

*Chapter 2: Create Benchmark Charge*
*Roll up the raw data into hourly intervals, and multiply each interval by the associated hourly price. These hourly charges are summed for the billing period to create a single charge per customer. The hourly prices are assumed to be available and error free.*

*Chapter 3: Output Benchmark Charge to Standard Format*
*Output the generated charge into a simple file format. One file is generated per customer.*

Breaking the bootstrap story into chapters turned out to have an unexpected result for our project. When the development team reviewed the chapters with the customer in order to explain how we were going to manage the large bootstrap story, the customer began to see some business value in individual chapters. The customer reached the conclusion that the chapters could be promoted to individual stories if they were reorganized slightly. The customer then re-prioritized the new stories. We ended up with three stories in place of our original bootstrap story:

*Story 1: Rollup 15 Minute Interval Data into 1 Hour Intervals*

*Load raw data from input files into the data warehouse. The data is assumed to be error free, so no data validation is required. Roll up the raw data into hourly intervals.*

This is a combination of chapter 1 and part of chapter 2. The significant realization by the customer was that there is business value in rolling up the raw data into hourly intervals. Because the costing algorithm was so simple, it could be done manually as long as the hourly interval data was available. This story became the minimum acceptable functionality for the first release, i.e. it became the new bootstrap story for the system. This new bootstrap story did not have any of the problems that the original one did because it is a more manageable size and has a more focused goal.

*Story 2: Create Benchmark Charge*
*Multiply each interval by the associated hourly price. These hourly charges are summed for the billing period to create a single charge per customer. The hourly prices are assumed to be available and error free.*

This is the same as chapter 2, with the exception that the rollup step was moved to story 1.

*Story 3: Output Benchmark Charge to Standard Format*
*Output the generated charge into a simple file format. One file is generated per customer.*

This is the same as chapter 3..

Each of these new stories fit within the iteration time box, so our iteration planning problems were resolved. We scheduled each story into sequential iterations. Our anarchistic task sign-up strategy was once again workable, because each story has its own set of cohesive tasks. Customer involvement at the functional level testing for each story was sufficient to cover all of the key processes. In our case, it made sense for the customer to adopt our chapters as stories. If they hadn't done so, the development team would have carried out our revised processes for the first release with the chapters we defined.

## 5 CONCLUSION

The story is the foundation for describing, planning, and managing an XP project. While the only dogma in XP is that there is no dogma, XP projects should endeavor to define and manage stories according to several simple guidelines:

- A story has business value to the customer.
- A story is prioritized by the customer.
- The tests for a story are defined by the customer.
- A story must be implemented entirely within an iteration.

XP teams should strive to not treat large stories, like the bootstrap story, as special cases. The process of simplifying and refactoring large stories opens up a dialog that often brings more clarity of the problem space to both the customer and the development team. Thinking creatively outside of the traditional boundaries of business processes will help shed light on where the true business value of the system lies. When all else fails, the last resort is to 'cheat' in order to remove barriers to success. We should not discourage cheating in XP, because after all the goal is to deliver a high quality working system, not to be a slave to set of principles and practices which merely guide us and keep us on track.

## ACKNOWLEDGEMENTS

The author would like to thank all the ClearStream colleagues who shared their experiences and insights in managing stories on a variety of XP projects. Much appreciation goes to Gerard Meszaros and Shaun Smith, who provided insightful feedback on earlier drafts of this paper. Thanks go out especially to the clients whose systems provided the opportunities for these experiences.

## REFERENCES

1. Beck, Kent. Extreme Programming Explained: Embrace Change, Addison-Wesley, 2000; ISBN 201-61641-6.
2. Beck, Kent. Martin Fowler, Planning Extreme Programming, Addison-Wesley, 2001; ISBN 0-201-71091-9.