Adventure XP

Report describing the AdventureXP project

Table of Contents

1 Introduction

For our AdventureXP project, we elected to follow the eXtreme Programming (XP) development methodology. This is a development methodology characterized by short development cycles and frequent, incremental releases. The goal is to increase communication and feedback from the customer, and develop software in such a way that the cost of changes in requirements is kept low. We felt this would prove to be an effective method for developing the AdventureXP system - a booking and inventory management system for an event planning company. Moreover, we also decided to implement some artifacts from Scrum as we felt it went hand-in-hand with XP.

2 How we used XP Artifacts

In the following section you will be able to read about what XP artifacts we used through this AdventureXP project as well as how we used said artifacts.

The underlying philosophy of XP is to take the best of software engineering practices and to bring them to a new high "extreme" level.

2.1 System Metaphor

Early in the development of the AdventureXP software we agreed to come up with a phrase that could describe the software under design in as few words as possible. We did this to have a mutual understanding of what we needed to develop as well as a common vocabulary that we could use when talking to our customer as well.

The system metaphor (which is not really a metaphor) that we came up with is the following:

"The system manages reservations for a group activity organization, and also keeps track of the inventory of equipment needed for those activities."

We decided to elaborate on the above and came up with the following longer paragraph. We did this to have an even better common vocabulary.

"Our system is a reservation and inventory system for AdventureXP, and as such, we communicate with the terms Reservation, Activity, Inventory and Items.

A Reservation contains information about an Activity at a specific time, with a group of people with a specific name.

An Activity is an event hosted by AdventureXP, like Go-Karting or Paintball. An Activity also has some collection of Items that must be available to make the Activity possible.

Inventory is the collection of Equipment, which are property of AdventureXP and must be available for some sort of Activity. Our system allows for the creation of reservations, as well as the management of the inventory required to enable the activities to fulfill those reservations."

2.2 Pair Programming

Pair programming is an agile software development technique, where two developers are working side-by-side on one computer. One is writing the code and the other helps him with suggestions, ideas and navigation. Moreover, the developers switch their roles frequently for more efficient results. This approach has a lot benefits like learning, design quality and develop of teamwork and communication.

We used this approach in our AdventureXP-project and it covered our expectations. We were able to work more efficient and we had the reward of more than one idea. Also, it was possible to reduce the errors and to follow the standards of code writing and design stricter.

Having said that, we also had times where pair programming was simply not the best choice and where individuals coded by themselves. Moreover, due to varying technical skills of the team members we also had, from time to time, teams of three programmers, where the "expert" would share his knowledge with the others.

2.3 Stand-up meetings

Stand-up meetings were used at the start of every "workday" to catch up on what each team member had been during on the previous "workday". Everybody shared how far along he or she was in the progress of completing their tasks/stories, what problems they had moving forward, and how much they expected to finish the given day.

The purpose of standing up whilst having the meeting was to keep it short and avoid long discussions.

2.4 Roles

Customer

The customer is the head of the project. She defines the project and sets its goals. She works closely with the developers, writes user stories and participates in the planning game.

In our case the customer was present during the planning game and when we had questions for the user stories. This part worked surprisingly well, and saved us from doing something wrong. Had there not been direct contact with the customer, we probably would have generated a lot of extra work, because some things would have been changed.

Developers

The developer's role in the project is being the expert and creating/maintaining the system. Developers work closely with the customer through the whole project. They estimate the amount of work on each user story, creates task from each user story and questions the stories if something could be done differently.

During the Adventure XP project, we - as developers - also worked closely with our customer. We estimated all user stories when doing the planning game, which went fairly well. There were almost no backlog. As mentioned in the role of a customer, the constant feedback saved us from a lot of misunderstandings.

Tracker

The tracker makes sure, that the project is heading in the right direction. He calculates team velocity and keeps track of any changes in the expected velocity. The tracker is valuable because of his ability to tell how changes might affect the schedule.

Morten functioned as our tracker during Adventure XP. He calculated our velocity, when we were placing our total amount of points to the given user stories.

Coach

The coach functions as a guidance and mentor for the team. He has a lot of knowledge and experience within XP and software development. He either teaches his team about certain parts of XP, or gives advices/suggestions to changes, implementations or technical issues.

In Adventure XP we did not have the pleasure of using a coach. Our group did contain some clever members though, who gave advice when we ran into "trouble".

2.5 Planning Game & Velocity

One of the core ideas of XP is the Planning Game. The Planning Game is a method of requirements gathering that engages the customer with the developers. The customer communicates their requirements to the developers, and the developers assign a cost estimate for a given feature. The customer may then pick and choose the highest priority requirements and purchase them with the hours of developer time they have for that week. In this way, the customer and the developers can work together to create the greatest value in the shortest period of time.

In the AdventureXP project, our customer communicated her requirements to us via user stories. A User Story is a means of defining requirements in a concise and direct way. A User Story conveys who the user is (“As a”), what they would like to do (“I want to”), and why (“so that”). This simple and direct way of communicating requirements allowed us to quickly understand what the customer wanted, and formulate questions to further define the specifics as necessary.

We used the Planning Game to gather the requirements for the AdventureXP project, and gave our customer hours with which to buy her desired functionality. In order to determine how many hours our customer could use per iteration in the purchasing process, we had to calculate our Project Velocity.

We had agreed to meet at specified hours, and so we knew how many hours each developer had per week. We took the sum of these man-hours, and subtracted any hours that we knew would have to be spent elsewhere. For example, if we had a piece of functionality that failed the acceptance test, we allocated time to correct that failure in the next iteration, and subtracted that from the hours available to the customer for the next iteration. Moreover, we also multiplied the total amount of hours by a percentage (e.g. 60%) as one does not work at 100% at all times.

<u>An example of hours to allocate could be:</u>

*- 8 people * 7 hours*	*= 56 hours*	
*- 56 hours * 60%*	*~ 33 hours*	*<u>33.5 hours to use for 1 week.</u>*

2.6 Style and Conventions

At the start of the project, we took some time to come to a consensus on how we would develop the software. We agreed on a set of coding style guidelines, so that our code would look consistent. We examined the AdventureXP project and developed a consistent System Metaphor with a precise vocabulary, defining terms like Activity, EventPackage and Member, in order to facilitate communication about the domain. We also agreed upon our development process; we were in agreement that we should use the Extreme Programming methodology, and follow its guidelines.

This meant that our development cycle would consist of a short planning phase, involving the entire team, and then any given task would be delegated to a two or three man team. Within the team, the workflow would consist of writing tests first, then writing code to fulfill the tests, following the Test Driven Development methodology. From there, after passing an Acceptance Test, code would be refactored, if necessary. Code for any given task could then be merged into the mainline.

2.7 The War-Room

In XP a War Room is a room in which people can work together as a team. Not only the developers should be present but everyone from the team as well as the customer. Moreover, the room should be wide and open and have whiteboards on as many walls as possible. By having a room like this discussions are encouraged and items such as User Stories and Story Boards are not lost, since they do not leave the room.

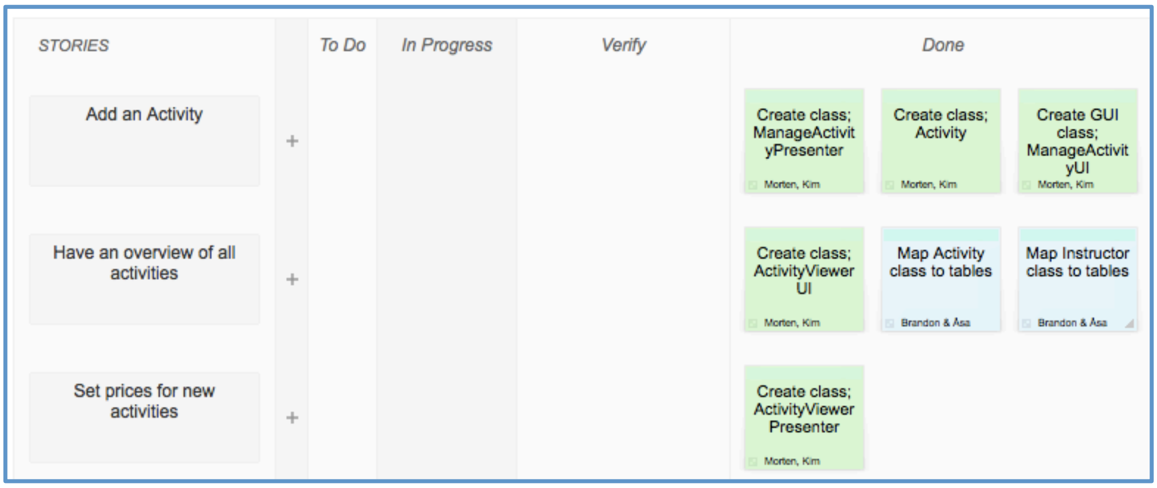
Throughout the AdventureXP project we have used this artifact and have constructed what we would call a War Room. All work that we have done throughout the project have been done in this room and we have used whiteboards as well as post-it’s and user story-cards to share ideas and thoughts. Moreover, the customer has also been present in this room so that if we had any questions/misunderstandings we could quickly get them out of the way.

3 How we used Scrum Artifacts

3.1 Story board

For keeping track of our User Stories and the tasks within each of these we decided to use a freeware called Scrumy (http://scrumy.com) for constructing a Story Board. We found this software of great use as we could interact with the Story Board from any computer as well as make changes at the same time. Moreover, the Story Board also worked as a motivating factor as more and more tasks were moved to the "Done" column.

In the figure below you can see a snapshot of our Story Board using Scrumy. The User Stories are shown to the left and the tasks within each User Story are shown to the right.



STORIES	To Do	In Progress	Verify	Done		
Add an Activity				Create class; ManageActivityPresenter (Morten, Kim)	Create class; Activity (Morten, Kim)	Create GUI class; ManageActivityUI (Morten, Kim)
Have an overview of all activities				Create class; ActivityViewer UI (Morten, Kim)	Map Activity class to tables (Brandon & Åsa)	Map Instructor class to tables (Brandon & Åsa)
Set prices for new activities				Create class; ActivityViewer Presenter (Morten, Kim)		

Figure 1: Story Board using Scrumy.com

3.2 Spikes

A spike is a fairly short time-boxed period of time used to research a concept and create a prototype for it. These prototypes are 'throw-away' meaning that they will not (necessarily) be used in the final release. One uses spikes to figure out if something is possible and to get rid of misunderstanding and difficulties.

We used spikes a couple of times throughout the AdventureXP project. The first one was when we agreed on using Hibernate for the construction of our database. No one had been working with Hibernate before and, therefore, we agreed on assigning the research and prototyping of said database to one of our team members.

4 Iteration 1 (2 weeks)

For the first iteration we had no prior experience with XP nor had most of us been working together as a team before. For that reason, we had a hard time calculating our velocity and we ended up having quite a few problems and unfinished User Stories for iteration 1. In the following section we will describe what went on during the first iteration

Having said that, we planned/had hoped to finish the following in iteration one.

4.1 Plan

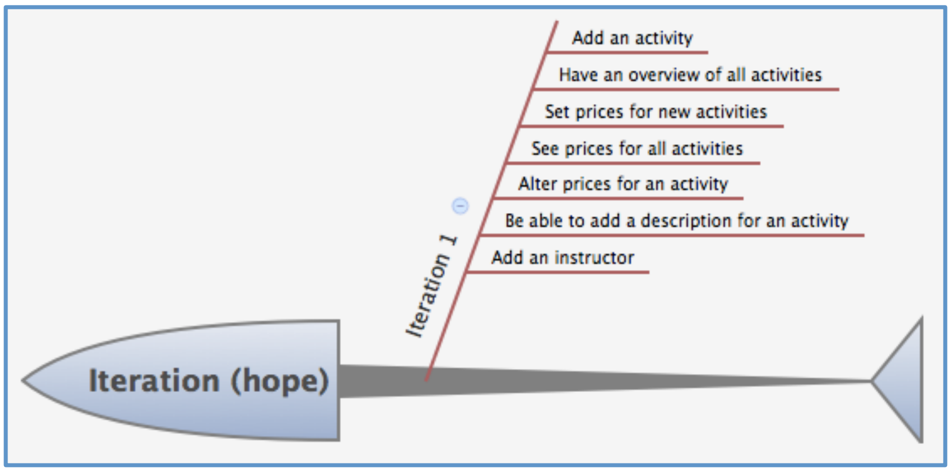

Iteration (hope)
Iteration 1
Add an activity
Have an overview of all activities
Set prices for new activities
See prices for all activities
Alter prices for an activity
Be able to add a description for an activity
Add an instructor

Figure 2: Plan for Iteration 1

4.2 Velocity

As previously said, we had a few difficulties when it came to calculating the velocity, as we did not know how efficient we could work. Therefore, we made a guesstimate and came up with the following:

• *1 Monday with 2 hours*			
• *2 Tuesdays with 5 hours*	*= 12 hours per person*		
• *8 people in the group*	*= 8 people * 12 hours*	*= 96 hours*	
• *Efficiency of 60%*	*= 96 * 0.6*	*= 57.6 hours*	*~ 57 hours*

We decided to put aside a total of 7 hours for things that did not have to do with user stories, so in conclusion, the customer had a total of 50 hours to spend on her user stories.

4.3 User stories (sorted by priority) & Tasks

User Story	
As a(n): Employee	**Priority:** 1
I want: Add an activity	**Size:** 10
So that: It is easy to have new activities	**Actual Size:** 14

- Create the following classes: ManageActivityPresenter, Activity, ManageActivityUI

User Story	
As a(n): Customer	**Priority:** 2
I want: Have an overview of all activities	**Size:** 10
So that: I know what I can choose from and see information about each activity	**Actual Size:** 10

- Create the following classes: ActivityViewerUI;

- Map the following classes to tables: Activity, Instructor

User Story	
As a(n): Employee	**Priority:** 3
I want: Set prices for new activities	**Size:** 4
So that: The price for hour is available	**Actual Size:** 1

- Create the following classes: ActivityViewerPresenter

User Story	
As a(n): Customer	**Priority:** 4
I want: See prices for all activities	**Size:** 10
So that: I know how expensive it is	**Actual Size:** 1

- Make relation between the following classes: Activity, ManageActivityUI

User Story	
As a(n): Employee	**Priority:** 5
I want: Be able to add a description for an activity	**Size:** 2
So that: It is easy to see the content of the activity and how long the activity normally last and if there are certain rules.	**Actual Size:** 2

- Add changes to the following classes: ManageActivityPresenter, Activity, ManageActivityUI

User Story	
As a(n): Employee	**Priority:** 6
I want: Add instructor	**Size:** 10
So that: I have details about the instructor such as name, address and phone.	**Actual Size:** 15

- Create the following classes: InstructorController, ManageInstructorPresenter, Instructor, ManageInstructorUI

User Story	
As a(n): Employee	**Priority:** 7
I want: Alter prices for an activity	**Size:** 4
So that: I can lower or rise prices	**Actual Size:** 4

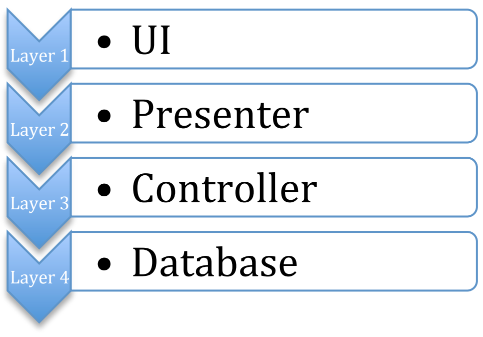
- Make relation between the following classes: ManageActivityPresenter, Activity

Note: Although we miscalculated some of the sizes for the User Stories we still ended up having done (most) of all User Stories. There were minor problems with some of them, which we ended up fixing in the beginning of Iteration 2.

Note 2: Some team members ended up spending more time than planned while others spent less. This was due to sickness and absence.

4.4 Logical architecture

During this Iteration we decided to use the following architecture for our application:



4.5 Problems encountered

During this Iteration we did not encounter many problems, but we did run in to a few. These are listed below.

- Get GitHub up and running on every workstation
- Figure out what kind of database to use for the project and get it working on every workstation
- Splitting up work in teams of two (some had more technical skills than others, so we)

5 Iteration 2 (2 weeks)

With the success of Iteration 1 we were ready for Iteration 2 and had a good feeling about the allocated Velocity from previous Iteration. However, during this iteration we ran in to a lot of unforeseen problems that caused some of the User Stories to be delayed.

We spent the start of the Iteration fixing minor bugs from Iteration 1. Therefore, the velocity for this iteration does not include the Tuesday where we got the new requirements.

5.1 Plan

We drafted the following plan, which has since been edited to match the unforeseen changes. The User Stories that are strikethrough have been moved till the next iteration.

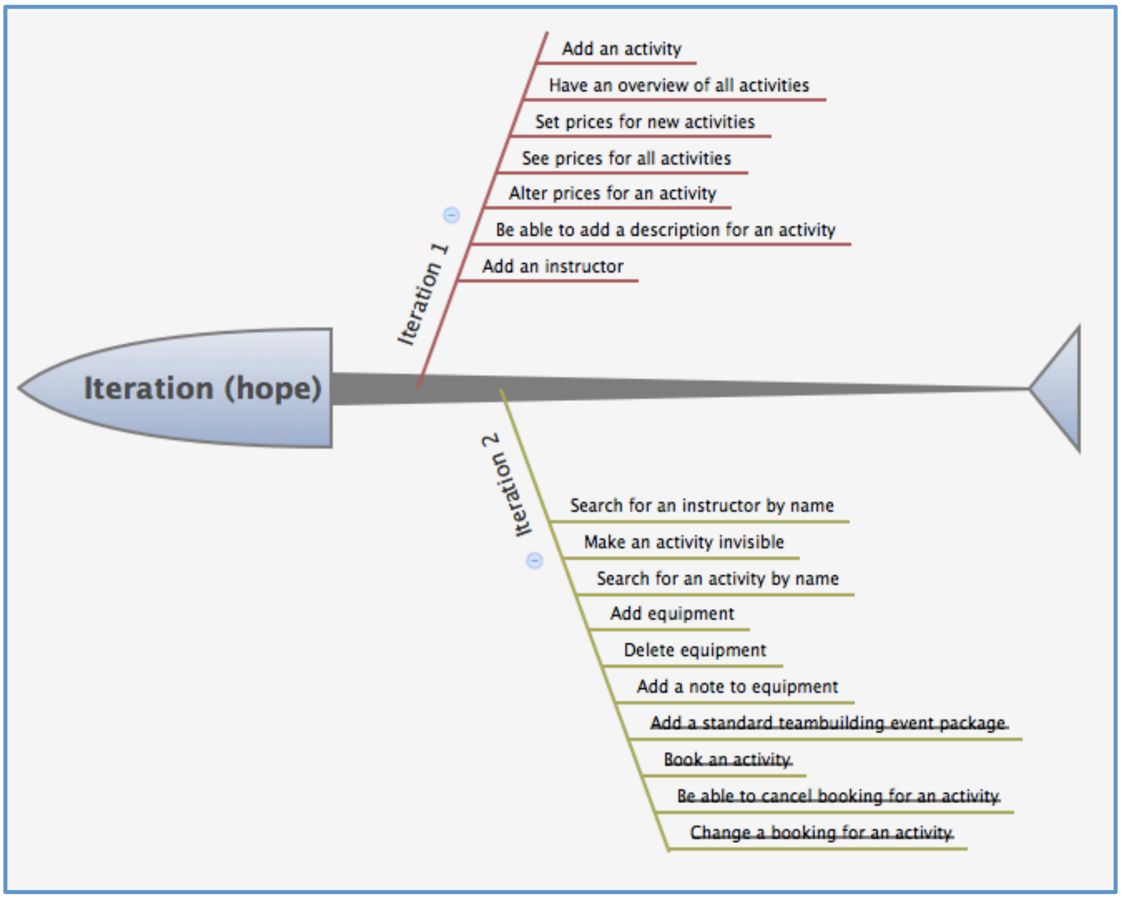

Iteration (hope)
Iteration 1
Add an activity
Have an overview of all activities
Set prices for new activities
See prices for all activities
Alter prices for an activity
Be able to add a description for an activity
Add an instructor
Iteration 2
Search for an instructor by name
Make an activity invisible
Search for an activity by name
Add equipment
Delete equipment
Add a note to equipment
~~Add a standard teambuilding event package~~
~~Book an activity~~
~~Be able to cancel booking for an activity~~
~~Change a booking for an activity~~

Figure 3: Plan for Iteration 2 (Strikethrough = Moved to next iteration)

5.2 Velocity

As previously mentioned, we decided to use the same schema for allocating hours for this iteration. However, we added a few more hours to the

- *2 Mondays with 4 hours*
- *1 Tuesday with 5 hours* *= 13 hours per person*

- *8 people in the group* *= 8 people * 13 hours* *= 104 hours*
- *Efficiency of 60%* *= 104 * 0.6* *~ 62 hours*

5.3 User stories (sorted by priority) & Tasks

User Story	
As a(n): Customer	**Priority:** 1
I want: To search for an activity by name	**Size:** 2
So that: I easily can all the information about this activity	**Actual Size:** 2

- Add changes in the following classes: ActivityController, ActivityViewerPresenter and ActivityViewerUI

User Story	
As a(n): Instructor	**Priority:** 2
I want: Add equipment	**Size:** 6
So that: I can have overview of the equipment	**Actual Size:** 10

- Add changes in the following classes: EquipmentController, EquipmentViewerPresenter, EquipmentViewerUI

User Story	
As a(n): Instructor	**Priority:** 3
I want: Delete equipment	**Size:** 2
So that: Missing or worn out equipment do not appear in the system.	**Actual Size:** 1

- Add changes in the following classes: EquipmentController, EquipmentViewerPresenter, EquipmentViewerUI

User Story	
As a(n): Employee	**Priority:** 4
I want: Search for an instructor by name	**Size:** 10
So that: I easily can see information about this instructor	**Actual Size:** 7

- Add changes to the following classes: InstructorController, InstructorViewerPresenter and InstructorViewerUI;

User Story	
As a(n): Instructor	**Priority:** 5
I want: Be able to add a note to equipment	**Size:** 2
So that: I can write if the equipment needs a repair.	**Actual Size:** 2

- Add changes in the following classes: EquipmentViewerPresenter, EquipmentViewerUI

User Story	
As a(n): Employee	**Priority:** 6
I want: Make an activity invisible	**Size:** 6
So that: It cannot be booked	**Actual Size:** 6

- Add changes in the following classes: ActivityViewerPresenter and ActivityViewerUI

User Story (moved till next iteration)	
As a(n): Employee	**Priority:** Important (extra user story)
I want: Add a standard teambuilding event package. The package consists of 4 activities and a fixed duration.	**Size:** 2
So that: Business customers can choose this package	**Actual Size:** Unknown

- Create class EventPackage, EventPackageController, EventPackageViewerPresenter
- Add changes to the AdministratorUI

User Story (moved till next iteration)	
As a(n): Customer	**Priority:** 7
I want: To book an activity	**Size:** 15
So that: I know a specific date and time is reserved for that activity	**Actual Size:** Unknown

Create the following classes: ManageBookingUI, ManageBookingPresenter, Booking

User Story (moved till next iteration)	
As a(n): Customer	**Priority:** 8
I want: To be able to cancel booking for an activity	**Size:** 5
So that: I can change my plans	**Actual Size:** Unknown

- Make changes at the following classes: BookingController,ManageBookingPresenter

User Story (moved till next iteration)	
As a(n): Customer/Employee	**Priority:** 9
I want: To be able to change booking for an activity	**Size:** 20
So that: The activity will take place at another date and time.	**Actual Size:** Unknown

Make changes at the following classes: BookingController, ManageBookingPresenter

5.4 Problems encountered

During this iteration we ran into problems when it came to creating bookings. We ended up spending a lot of hours working on this, but ended up having to move it to Iteration 3. Besides this everything ran to plan, although some team members were missing on workdays.

6 Iteration 3 (1 week)

For the third iteration, we had minor tasks left over from the second iteration to complete, but they took relatively little time out of the iteration. The rest of the time allocated for Iteration 3 was spent on developing the functionality for booking activities, which was highly prioritized by the client.

However, we did not manage finish this functionality.

6.1 Plan

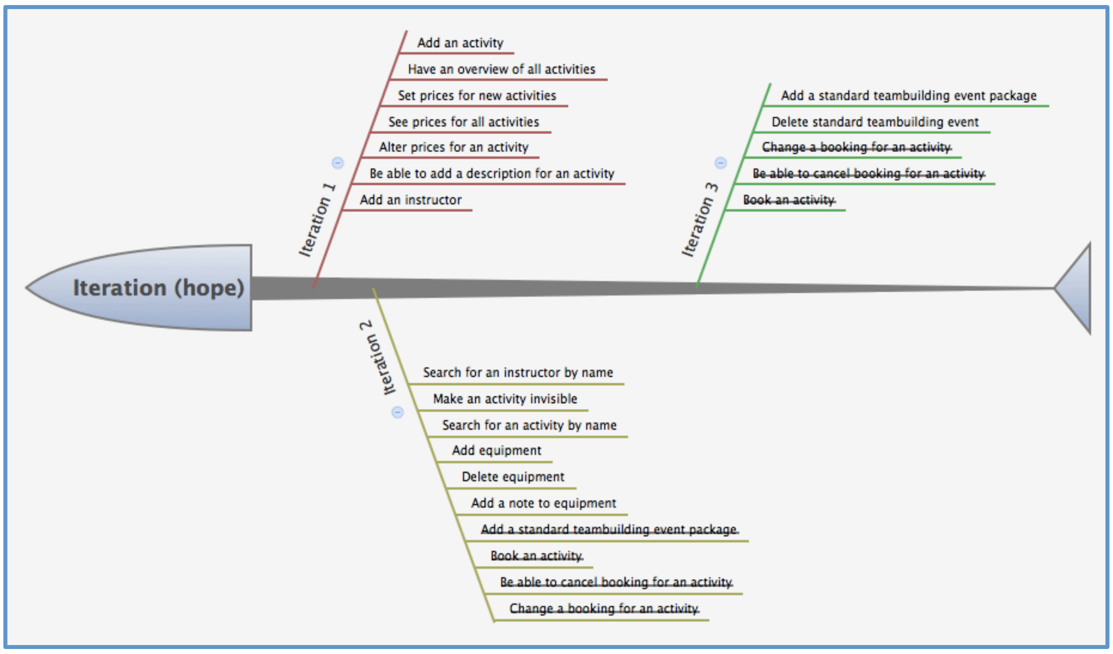

Iteration (hope)
Iteration 1
Add an activity
Have an overview of all activities
Set prices for new activities
See prices for all activities
Alter prices for an activity
Be able to add a description for an activity
Add an instructor
Iteration 3
Add a standard teambuilding event package
Delete standard teambuilding event
~~Change a booking for an activity~~
~~Be able to cancel booking for an activity~~
~~Book an activity~~
Iteration 2
Search for an instructor by name
Make an activity invisible
Search for an activity by name
Add equipment
Delete equipment
Add a note to equipment
~~Add a standard teambuilding event package~~
~~Book an activity~~
~~Be able to cancel booking for an activity~~
~~Change a booking for an activity~~

Figure 4: Iteration plan for Iteration 3 (Strikethrough = Moved to next iteration)

6.2 Velocity

For this iteration we allocated approximately 20 hours as most team members had other things to attend.

6.3 User stories (sorted by priority) & Tasks

User Story	
As a(n): Customer	**Priority:** 1
I want: To book an activity	**Size:** 25
So that: I know a specific date and time is reserved for that activity	**Actual Size:** Unknown

Create the following classes: ManageBookingUI, ManageBookingPresenter, Booking

User Story	
As a(n): Customer	**Priority:** 2
I want: To be able to cancel booking for an activity	**Size:** 5
So that: I can change my plans	**Actual Size:** Unknown

- Make changes at the following classes: BookingController,ManageBookingPresenter

User Story	
As a(n): Customer/Employee	**Priority:** 3
I want: To be able to change booking for an activity	**Size:** 26
So that: The activity will take place at another	**Actual Size:** Unknown

date and time.

Make changes at the following classes: BookingController, ManageBookingPresenter

User Story (moved till next iteration)	
As a(n): Employee	**Priority:** Important (extra user story)
I want: Add a standard teambuilding event package. The package consists of 4 activities and a fixed duration.	**Size:** 2
So that: Business customers can choose this package	**Actual Size:** 2

- Create class EventPackage, EventPackageController, EventPackageViewerPresenter
- Add changes to the AdministratorUI

User Story	
As a(n): Employee	**Priority:** 4
I want: Delete standard teambuilding event	**Size:** 2
So that: It cannot be booked	**Actual Size:** 2

Create the following classes : EventPackage, ManageEventPackagePresenter, EventpackageController

6.4 Problems encountered

Iteration 3 was challenging because it was a shorter iteration that demanded a relatively complex piece of functionality. While we calculated for this, and had little aside from those user stories to complete, we were shorter on developers that week, and in the end, the booking functionality was not entirely completed.

We also ran into a few problems with our database communication layer (Hibernate), that we did not detect due to flaws in some of our Unit Tests. The problem wasn't immediately visible in normal operation of the program, so we detected it late, and this had to be fixed.

So, while we were not successful in delivering a high priority feature in Iteration 3, the software itself was still greatly improved.

7 Conclusion

We feel that we all learned a lot in the process of developing the AdventureXP project, and it gave us insight into the Extreme Programming methodology, which was a new way of developing software for most of us.

While we were unable to always complete all user stories in the iteration we wanted, there was never a point that we felt completely overwhelmed. The process allowed for a great deal of flexibility, and our priorities were always very clear. This meant that very little time went to waste.

Using test driven development meant that we always had a suite of tests to check for regressions in the code base, and though we were inexperienced with this style of developing, we felt that in the end it was worth it, as the unit tests allowed us to very frequently merge and integrate our code base, and quickly resolve any errors that arose.

Overall, it was a very useful learning experience.