

# Extreme Programming

## Driving Metaphor:

*Driving a car is not about pointing the car in one direction and holding to it; driving is about making lots of little course corrections.*

*"Do the simplest thing that could possibly work"*

## A set of interacting practices:

Planning Game	Testing	Continuous Integration
Short Releases	Refactoring	40-Hour-Week
Metaphor	Pair Programming	On-Site Customer
Simple Design	Collective Code Ownership	Coding Standards

## Why we plan

### We want to ensure that

- ☐ *we are always working on the most important things*
- ☐ *we are coordinated with other people*
- ☐ *when unexpected events occur, we understand the consequences on priorities and coordination*

### Plans must be

- ☐ *easy to make and update*
- ☐ *understandable by everyone that uses them*

## *The Planning Trap*

- ❑ *Plans project a likely course of events*
- ❑ *Plans must try to create visibility: where is the project*

**But:** A plan does not mean you are in control of things

- ❑ Events happen
- ❑ Plans become invalid

**Problem:** Fear of being blamed induces planner to say that the project is still on track

☞ *Divergence between plan and reality*

*Having a plan isn't everything, planning is.*

☞ *Keep plans honest and expect them to always change*

## Business - Development Relationships

**A well-known experience in Software Development:** *The customer and the developer sit in a small boat in the ocean and are afraid of each other.*

Business fears:	Development fears:
<i>They won't get what they asked for</i>	<i>They won't be given clear definitions of what needs to be done</i>
<i>They must surrender the control of their careers to techies who don't care</i>	<i>They will be given responsibility without authority</i>
<i>They'll pay too much for too little</i>	<i>They will be told to do things that don't make sense</i>
<i>They won't know what is going on (the plans they see will be fairy tales)</i>	<i>They'll have to sacrifice quality for deadlines</i>

**Result:** *A lot of energy goes into protective measures and politics instead of success*



*XP: We must create a culture that makes it possible for programmers and customers to acknowledge their fears and accept their rights and responsibilities.*

## ***The Manager and Customer Bill of Rights***

- ❑ *You have the right to an overall plan*
  - *To steer a project, you need to know what can be accomplished within time and budget*
- ❑ *You have the right to get the most possible value out of every programming week*
  - *The most valuable things are worked on first.*
- ❑ *You have the right to see progress in a running system.*
  - *Only a running system can give exact information about project state*
- ❑ *You have the right to change your mind, to substitute functionality and to change priorities without exorbitant costs.*
  - *Market and business requirements change. We have to allow change.*
- ❑ *You have the right to be informed about schedule changes, in time to choose how to reduce the scope to restore the original date.*
  - *XP works to be sure everyone knows just what is really happening.*

# The Programmer Bill of Rights

- ❑ You have the right to know what is needed, with clear declarations of priority.
  - Tight communication with the customer. Customer directs by value.
- ❑ You have the right to produce quality work all the time.
  - Unit Tests and Refactoring help to keep the code clean
- ❑ You have the right to ask for and receive help from peers, managers, and customers
  - No one can ever refuse help to a team member
- ❑ You have the right to make and update your own estimates.
  - Programmers know best how long it is going to take them
- ❑ You have the right to accept your responsibilities instead having them assigned to you
  - We work most effectively when we have accepted our responsibilities instead of having them thrust upon us

## Balancing Power

**XP process enforces the separations of roles:**

- ❑ *Business people make business decisions*
- ❑ *Technical people make technical decisions*

Business Decisions	Technical Decisions
<i>Scope</i>	<i>Estimates</i>
<i>Dates of the releases</i>	<i>Dates within an iteration</i>
<i>Priority</i>	<i>Team velocity</i>
	<i>Warnings about technical risks</i>

*The Customer owns "what you get" while the Programmer owns "what it costs".*

## Requirements for the XP Customer

*In XP, the customer plays an essential role: she steers the project.*

*A good customer*

- ☐ *understands the domain well by working in that domain, and also by understanding how it works*
- ☐ *can understand, with developments help, how software can provide business value within the domain*
- ☐ *can make decisions about what's needed now and what's needed later*
- ☐ *is willing to accept ultimate responsibility for the success or failure of the project*

*If you get lost driving, it's not the car's fault, it's the driver's!*



# The Planning Game

*A game with a set of rules that ensures that business and development don't become mortal enemies*

## **Goal:**

*Maximize the value of the software produced by the development team.*

## **Overview:**

**Release Planning:** *Business selects the scope of the next release*

**Iteration Planning:** *Development decides on what to do and in which order*

# The Release Planning Game

	Business	Development
<i>Exploration Phase</i>	<i>Write Story</i>	
		<i>Estimate Story</i>
	<i>Split Story</i>	
<i>Commitment Phase</i>	<i>Sort Stories by Value</i>	
		<i>Sort Stories by Risk</i>
		<i>Set Velocity</i>
	<i>Choose Scope</i>	
<i>Steering Phase</i>	<i>Iteration</i>	
		<i>Recovery</i>
	<i>New Story</i>	<i>Reestimate</i>

# Planning Game: Exploration Phase

## Purpose:

*Give both players an appreciation for what all the system should eventually do.*

## The Moves:

- ☐ **Customer:** *Write a story. Discuss it until everybody understands it.*
- ☐ **Programmer:** *Estimate a story in terms of effort.*
- ☐ **Customer:** *Split a story, if programmers don't understand or can't estimate it.*
- ☐ **Programmer:** *Do a spike solution to enable estimation.*
- ☐ **Customer:** *Toss Stories that are no longer wanted or are covered by a split story.*

# User Stories


*A story is the unit of functionality in an XP project. It is the medium to do analysis in XP.*

## **Principles of good stories:**

- ❑ *Stories must be understandable to the customer*
  - ☞ *use plain, natural english/german/kishuaeli (no formalized language)*
- ❑ *Customers write stories. Developers do *not* write stories.*
  - ☞ *All proposals by developers are automatically approved by the customer*
- ❑ *The shorter the better. No detailed specification!*
  - ☞ *A story is nothing more than an agreement that the customer and the developer will talk together about a feature.*
- ❑ *Each story must provide something of value to the customer*
  - ☞ *then we avoid building fancy gadgets that will not be used*
- ❑ *A Story must be testable*
  - ☞ *then we can know when it is done*

# Writing User Stories

*Writing stories is an iterative process, requiring lots of feedback.*

- Customers will propose a story to the programmers*
- Programmers decide if story can be tested and estimated*
- Customers may have to clarify or split the story*
  
- ☐ *Write stories on index cards*
  - cards are simple, physical devices that invite everybody to manipulate them*
  
- ☐ *A story contains not more than a few sentences*
  - if the story is too big, write only the essential core*
  
- ☐ *If the story is not quite right, tear the card up and write a new one*
  -  *our mindset remains flexible*

# Sample Stories

A story contains:

- ☐ a name
- ☐ the story itself
- ☐ an estimate

## Story Examples:

- ☐ *When the GPS has contact with two or fewer satellites for more than 60 seconds, it should display the message "Poor satellite contact", and wait for confirmation from the user. If contact improves before confirmation, clear the message automatically.*
- ☐ *If the station currently playing carries digital information, the information is displayed in the radio's LCD display. If there is no digital information available, display the station frequency.*

# Splitting Stories

## **Programmers ask the customer to split a story if**

- ☐ *They cannot estimate a story because of its complexity*
- ☐ *Their estimate is longer than two or three weeks of effort*

## **Reasons:**

- ☐ *Estimates get fuzzy for bigger stories*
- ☐ *The smaller the story, the better the control (tight feedback loop)*

## **Some rules for splitting stories:**

- ☐ *Stories that have an important part and a less important part: make two stories*
- ☐ *Stories covering several related cases: make each case a story*

**Also:** *Bundle stories with estimates of less than a day together: avoid cluttering the plan*

## Use Cases vs. User Stories

***Use Case:** An action with multiple participant objects that represent a meaningful business task, usually written in a structured, narrative style. Can be refined into a finer grained sequence of actions. (Catalysis)*

### **Additional information can include:**

- ☐ *Pre- and postconditions*
- ☐ *Failure conditions, failure actions*
- ☐ *Refinement criteria: what to consider when refining this use case into a sequence*
- ☐ *Action sequence*
- ☐ *Frequency, performance, data descriptions of the in/out data*

### **Compared to User Stories, Use Cases are**

- ☐ *More precise:  
consistent, complete, non-overlapping, non-contradicting*
- ☐ *Hierarchically organized*
- ☐ *Quickly a part of 'Big Methodology'*




# Estimating Stories

## Keys to effective story estimation:

- ☐ *Keep it simple*
- ☐ *Use what happened in the past ("Yesterday's weather")*
- ☐ *Learn from experience*

## Comparative story estimation:

- ☐ *One story is often an elaboration of a closely related one*
- ☐ *Look for stories that have already been implemented*
- ☐ *Compare difficulties, not implementation time (Velocity!)*
  -  *"twice as difficult", "half as difficult"*
- ☐ *Discuss estimates in the team. Try to find an agreement.*
- ☐ *"Optimism wins": Choose the more optimistic of two disagreeing estimates.*

## **Initial Estimation of Stories**

*With no history, the first plan is the hardest and least accurate  
(Fortunately, you only have to do it once)*

### **How to start estimating:**

- ☐ *Begin with the stories that you feel the most comfortable estimating.*
- ☐ *Intuitively imagine how long it will take you.*
- ☐ *Base other estimates on the comparison with those first stories.*

### **Spike Solutions:**

*Do a quick implementation of the whole story.*

- Do not look for the perfect solution!*
- Just try to find out how long something takes*

# Planning Game: Commitment Phase I

## Purpose:

**Business:** *to choose scope and date of next delivery*

**Development:** *to confidently commit to deliver the next release*

## The Moves:

### ☐ **Business:** Sort by stories by value

(1) *Stories without which the system will not function*

(2) *Less essential stories, but still providing significant business value*

(3) *Nice-to-have stories*

☞ *Business wants the release to be as valuable as possible*

### ☐ **Development:** Sort stories by risk

(1) *Stories that can be estimated precisely (low risk)*

(2) *Stories that can be estimated reasonably well*

(3) *Stories that cannot be estimated at all (high risk)*

☞ *Development wants to tackle high-risk first, or at least make risk visible*

## Planning Game: Commitment Phase II

- ❑ **Development:** Set team velocity  
*How much ideal engineering time per calendar month/week can the team offer?*  
☞ *this is the budget that is available to Business*
  
- ❑ **Business:** Choose scope of the release, by either
  - *fixing the date and choosing stories based on estimates and velocity*
  - *fixing the stories and calculating the delivery date*

### Shopping Analogy:

*You have a budget for groceries for a week. If you have to feed a family of seven, you go for spaghetti. If the boss comes over for dinner, you get steaks and eat bread for the rest of the week.*

## Planning Game: Steering Phase

**Purpose:** *Update the plan based on what is learned.*

**The Moves:**

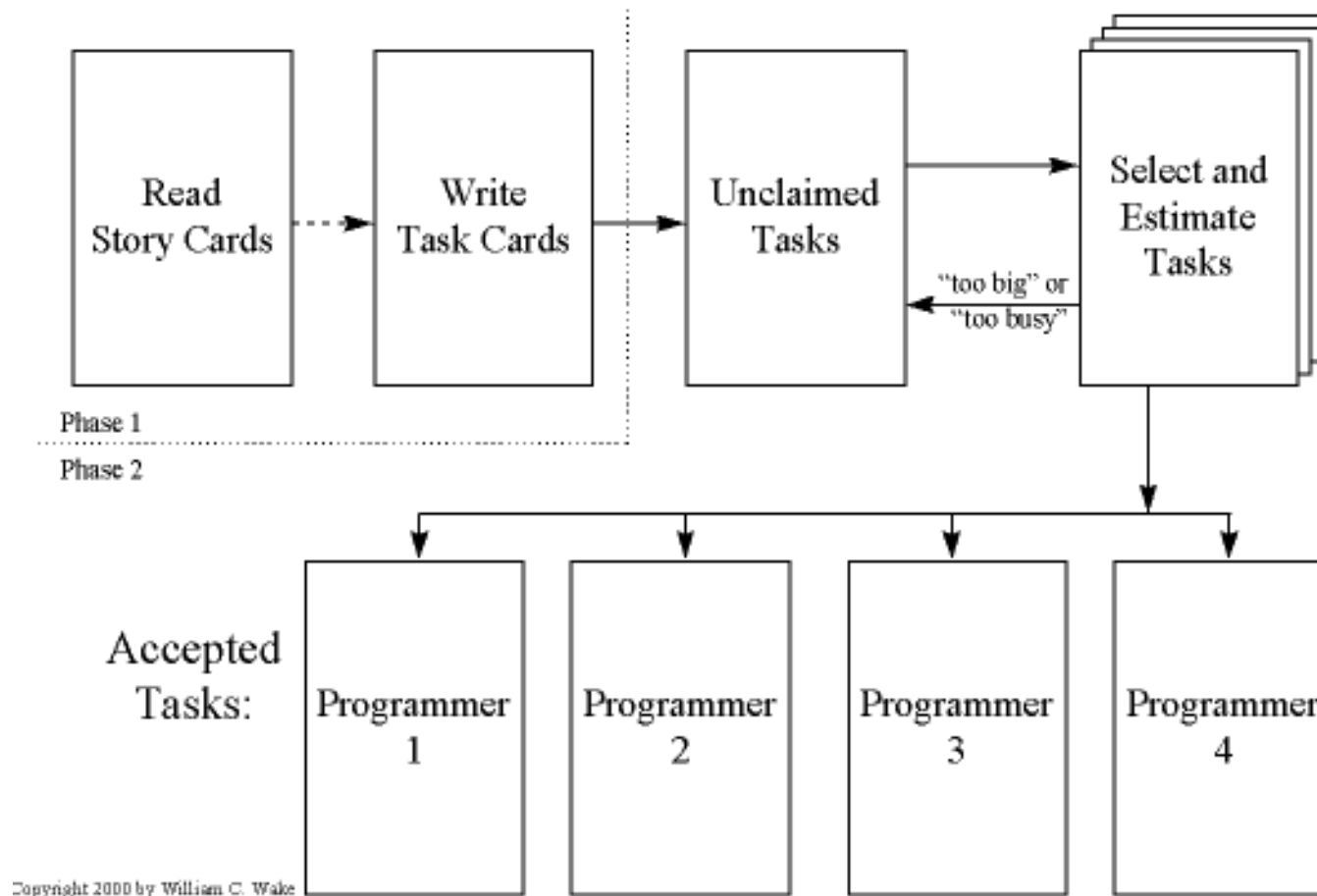
- ❑ **Iteration:** *Business picks one iteration worth of the most valuable stories.*
  - ☞ *see Iteration Planning*
- ❑ **Get stories done:** *Business should only accept stories that are 100% done.*
- ❑ **Recovery:** *Programmer realizes velocity is wrong*
  - *Development re-estimates velocity.*
  - *Business can defer (or split) stories to maintain release date.*
- ❑ **New Story:** *Business identifies new, more valuable story*
  - *Development estimates story*
  - *Business removes estimated points from incomplete part of existing plan, and inserts the new story.*
- ❑ **Reestimate:** *Development feels that plan is no longer accurate*
  - *Development re-estimates velocity and all stories.*
  - *Business sets new scope plan.*

# Iteration Planning I

- ❑ Customer selects stories to be implemented in this iteration.
- ❑ Customer explains the stories in detail to the development team
  - Resolve ambiguities and unclear parts in discussion
- ❑ Team brainstorms engineering tasks
  - A task is small enough that everybody fully understands it and can estimate it.
  - Use quick CRC or UML sessions to determine how a story is accomplished.
  - ☞ Observing the design process builds common knowledge and confidence throughout the team
- ❑ Programmers/Programmer Pairs sign up for work and estimates
  - Assignments are not forced upon anybody (Principle of Accepted Responsibility)
  - The person responsible for a task gets to do the estimate

# Iteration Planning II

## Overview of the Planning Meeting:



# Sample Engineering Task

## Story:

*Restrict access to the application. Only allow people who know a user's username and password to access ATS.*

## Tasks:

- ☐ *Add context column to user table. (Migrate data too)*
- ☐ *Perform login and logout. (SessionManager)*
- ☐ *Login/logout window.*
- ☐ *Create SecurityException class.*
- ☐ *Track current user's session on client.*
- ☐ *Dialog for login failure and bad session.*
- ☐ *Test: make sure password isn't cached.*
- ☐ *Functional tests for application access*