Introduction to Software Engineering

3. The Planning Game
Roadmap

> XP — coping with change and uncertainty
> Customers and Developers — why do we plan?
> The Planning Game
> Iteration
> Scrum
> Agile lessons from industry

Based on a presentation by Matthias Rieger.
Sources


> [www.extremeprogramming.org](http://www.extremeprogramming.org)
Roadmap

> XP — coping with change and uncertainty
> Customers and Developers — why do we plan?
> The Planning Game
> Iteration
> Scrum
> Agile lessons from industry
Extreme Programming

XP is a set of mutually supportive practices for developing quality software

- Testing
- Refactoring
- Pair programming
Extreme Programming

XP is a set of mutually supportive practices for developing quality software.

- Continuous Integration
- Testing
- Refactoring
- Pair programming
- Collective Code Ownership
- Coding Standards

© Joseph Pelrine
Extreme Programming

Simple Design
System Metaphor
Continuous Integration
Teamwork
Coding

Testing
Refactoring
Pair programming

Collective Code Ownership
Coding Standards

40 Hour Week

© Joseph Pelrine
Extreme Programming

Planning Game

Simple Design

System Metaphor

Continuous Integration

Product

Process

Teamwork

Coding

Testing

Refactoring

Pair programming

Collective Code Ownership

Coding Standards

40 Hour Week

On-Site Customer

Small Releases

See also: www.extremeprogramming.org

© Joseph Pelrine
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”
Roadmap

> XP — coping with change and uncertainty
> Customers and Developers — why do we plan?
> The Planning Game
> Iteration
> Scrum
> Agile lessons from industry
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

Having a *plan isn’t* everything, *planning* is.
— Keep plans honest and expect them to always change
Customer-Developer 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.

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

**Result:** A lot of energy goes into protective measures and politics instead of success
# The Customer Bill of Rights

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

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

**Customer** makes business decisions

**Developers** make technical decisions

<table>
<thead>
<tr>
<th>Business Decisions</th>
<th>Technical Decisions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope</td>
<td>Estimates</td>
</tr>
<tr>
<td>Dates of the releases</td>
<td>Dates within an iteration</td>
</tr>
<tr>
<td>Priority</td>
<td>Team velocity</td>
</tr>
<tr>
<td></td>
<td>Warnings about technical risks</td>
</tr>
</tbody>
</table>

*The Customer owns “what you get” while the Developers own “what it costs”.*
Roadmap

> XP — coping with change and uncertainty
> Customers and Developers — why do we plan?
> The Planning Game
  — Exploration — User stories
  — Estimation
  — Commitment
  — Steering
> Iteration
> Scrum
> Agile lessons from industry
The Planning Game

A game with a set of rules that ensures that Customer and Developers don’t become mortal enemies

Goal:
— Maximize the value of the software produced by Developers.

Overview:
1. Release Planning: Customer selects the scope of the next release
2. Iteration Planning: Developers decide on what to do and in which order
# The Release Planning Game

<table>
<thead>
<tr>
<th>Phase</th>
<th>Customer</th>
<th>Developers</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Exploration Phase</strong></td>
<td>Write Story</td>
<td>Estimate Story</td>
</tr>
<tr>
<td></td>
<td>Split Story</td>
<td></td>
</tr>
<tr>
<td><strong>Commitment Phase</strong></td>
<td>Sort Stories by Value</td>
<td>Sort Stories by Risk</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Set Velocity</td>
</tr>
<tr>
<td></td>
<td>Choose Scope</td>
<td></td>
</tr>
<tr>
<td><strong>Steering Phase</strong></td>
<td>Iteration</td>
<td>Recovery</td>
</tr>
<tr>
<td></td>
<td>New Story</td>
<td>Reestimate</td>
</tr>
</tbody>
</table>
Roadmap

> XP — coping with change and uncertainty
> Customers and Developers — why do we plan?
> The Planning Game
  — Exploration — User stories
  — Estimation
  — Commitment
  — Steering
> Iteration
> Scrum
> Agile lessons from industry
Planning Game: Exploration Phase

**Purpose:**

Get an appreciation for what the system should eventually do.

**The Moves:**

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

*Principles of good stories:*

> **Customers** write stories.
> — **Developers** do *not* write stories.

> Stories must be *understandable* to the customer

> The *shorter* the better. No detailed specification!
> — Write stories on *index cards*

> Each story must provide *something of value* to the customer

> A story must be *testable*
> — then we can know *when it is done*

*Writing stories is an iterative process, requiring interaction between Customer and Developers.*
Stories

A story contains:

> a name
> the story itself
> an estimate

Example:

—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.
Splitting Stories

*Developers 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

*Why?*

> Estimates get fuzzy for bigger stories
> The smaller the story, the better the control (tight feedback loop)
Roadmap

> XP — coping with change and uncertainty
> Customers and Developers — why do we plan?
> The Planning Game
   — Exploration — User stories
   — **Estimation**
   — Commitment
   — Steering
> Iteration
> Scrum
> Agile lessons from industry
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
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
  —“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.
Roadmap

- XP — coping with change and uncertainty
- Customers and Developers — why do we plan?
- The Planning Game
  — Exploration — User stories  
  — Estimation  
  — Commitment  
  — Steering
- Iteration
- Scrum
- Agile lessons from industry
Planning Game: Commitment Phase

**Purpose:**

> **Customer:** to choose scope and date of next delivery  
> **Developers:** to confidently commit to deliver the next release

**The Moves:**

> **Customer:** *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  
   — Customer wants the release to be as *valuable* as possible
Commitment Phase …

> **Developers:** 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 (*high risk*)
> — Developers want to tackle *high-risk first*, or at least make risk visible

> **Developers:** Set team *velocity*
> How much ideal engineering time per calendar month/week can the team offer?
> — this is the *budget* that is available to Customer

> **Customer:** 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
Roadmap

> XP — coping with change and uncertainty
> Customers and Developers — why do we plan?
> The Planning Game
  — Exploration — User stories
  — Estimation
  — Commitment
  — **Steering**
> Iteration
> Scrum
> Agile lessons from industry
Planning Game: Steering Phase

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

**The Moves:**

> *Iteration:* **Customer** picks one iteration worth of the most valuable stories.
  
  — see Iteration Planning

> *Get stories done:* **Customer** should only accept stories that are 100% done.

> *Recovery:* **Developers** realize velocity is wrong
  
  — **Developers** re-estimate velocity.
  
  — **Customer** can defer (or split) stories to maintain release date.
Planning Game: Steering Phase...

> New Story: Customer identifies new, more valuable stories
  — Developers estimate story
  — Customer removes estimated points from incomplete part of existing plan, and inserts the new story.

> Reestimate: Developers feel that plan is no longer accurate
  — Developers re-estimate velocity and all stories.
  — Customer sets new scope plan.
Roadmap

> XP — coping with change and uncertainty
> Customers and Developers — why do we plan?
> The Planning Game
  — Exploration — User stories
  — Estimation
  — Commitment
  — Steering
> Iteration
> Scrum
> Agile lessons from industry
Iteration Planning

An Iteration Planning Game

Read Story Cards → Write Task Cards → Unclaimed Tasks → Select and Estimate Tasks

"too big" or "too busy"

Accepted Tasks:

Programmer 1 → Programmer 2 → Programmer 3 → Programmer 4

© 2000 William C. Wake
Iteration Planning

> **Customer** selects stories to be implemented in this iteration.

> **Customer** explains the stories in detail to the Developers
  — Resolve ambiguities and unclear parts in discussion

> **Developers** brainstorm **engineering tasks**
  — A task is small enough that everybody fully understands it and can estimate it.
  — Use short CRC or UML sessions to determine how a story is accomplished.
  — Observing the design process builds common knowledge and confidence throughout the team

> **Developers /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
Roadmap

> XP — coping with change and uncertainty
> Customers and Developers — why do we plan?
> The Planning Game
  — Exploration — User stories
  — Estimation
  — Commitment
  — Steering
> Iteration
> Scrum
> Agile lessons from industry
Scrum is a framework of an agile process to manage software projects.

Product backlog → Sprint backlog → Sprint → Working increment of the software

24 h → 30 days
Scrum roles

1. **Scrum Master** — manages the process
   — removes impediments to the team
2. **Product owner** — stakeholder
   — prioritizes product backlog items
3. **Team** — ~7 developers (analysis, design etc)
   — decides which backlog items go into a sprint
Chickens and Pigs

> Pigs are *committed* to the project
   — Scrum Master, Product Owner, Team
   — Part of the Scrum process

> Chickens are only *involved*
   — Managers, Stakeholders (vendors, customers)
   — Should be taken into account
   — Not part of the process!
Daily Scrum (standup meeting)

- Same time, same place, every day
- Start on time
- Max 15"
- Only pigs may speak
- Answer 3 questions
  1. What have I done since yesterday?
  2. What am I planning to do today?
  3. What problems are preventing me from reaching my goal?
Other Scrum meetings

> **Scrum of scrums**
  — meeting of teams, after the daily scrum

> **Sprint planning**
  — prepare sprint backlog at start of sprint

> **Sprint review**
  — review work completed at end of sprint

> **Sprint retrospective**
  — review sprint process
Roadmap

> XP — coping with change and uncertainty
> Customers and Developers — why do we plan?
> The Planning Game
  — Exploration — User stories
  — Estimation
  — Commitment
  — Steering
> Iteration
> Scrum
> Agile lessons from industry
Agile Lessons from Industry

> Need dedicated team
> Willingness to prioritize requirements
> Engaged product manager
> Clear project governance
> Project team should shape the process to its needs
> Transparency and openness
> Project success measured by delivered value
Ingredients for Success

For the collaboration:
> Stakeholders must understand the process
> Need efficient decision making process
> Someone must translate technical ↔ business
> Team must agree on ground rules
> Plan time to assess and improve collaboration
Ingredients for Success

For planning reliability:

> Aim for reliable sprint planning
  — shorter is easier to estimate

> Define clearly what “complete” means

> Put project management and planning on the sprint backlog
Agile methods still need classical project management

> Leadership is needed to facilitate “self-organization”
> Frequent planning is required
> Structure and discipline are needed
> Continuous dialog with product owner takes time
> Not every phase of project can be “agiled”
  —e.g. product launch and move to operation
What you should know!

> Why is planning more important than having a plan?
> Why shouldn’t Customers make technical decisions? Why shouldn’t Developers make business decisions?
> Why should stories be written on index cards?
> Why should the Customer sort stories by value?
> Why should the Developer sort stories by risk?
> How do you assign stories to Developers?
Can you answer the following questions?

> What is “extreme” about XP?
> What is the differences between a User Story and a Use Case?
> Are Developers allowed to write stories?
> What is the ideal time period for one iteration?
> How can you improve your skill at estimating stories?
Attribution-ShareAlike 3.0

You are free:
- to copy, distribute, display, and perform the work
- to make derivative works
- to make commercial use of the work

Under the following conditions:

**Attribution.** You must attribute the work in the manner specified by the author or licensor.

**Share Alike.** If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

http://creativecommons.org/licenses/by-sa/3.0/