

# Terms Of Use

## Nutzungsbedingungen



- This slide set is subject to the “terms of use” as set out under <http://www.xpexchange.net/english/terms.html>
- Diese Präsentation unterliegt den “Nutzungsbedingungen”, wie sie zu finden sind unter: <http://www.xpexchange.net/german/terms.html>

# Customer Bill Of Rights



- You have the right to always know the actual status and progress of the project.
- You have the right to decide on WHAT gets implemented.
- You have the right to add, change, and remove requirements at any time.
- You have the right to get the most out of every programming week.
- You have the right to get a working system every 3 to 4 months.

# Programmers Bill Of Rights



- You have the right to decide on HOW things are implemented.
- You have the right to create systems with the highest quality possible.
- You have the right to ask the customer for clarification of requirements at any time.
- You have the right to estimate the effort for implementing a system.
- You have the right to change estimates based on new findings.



# A New Way of Developing Software

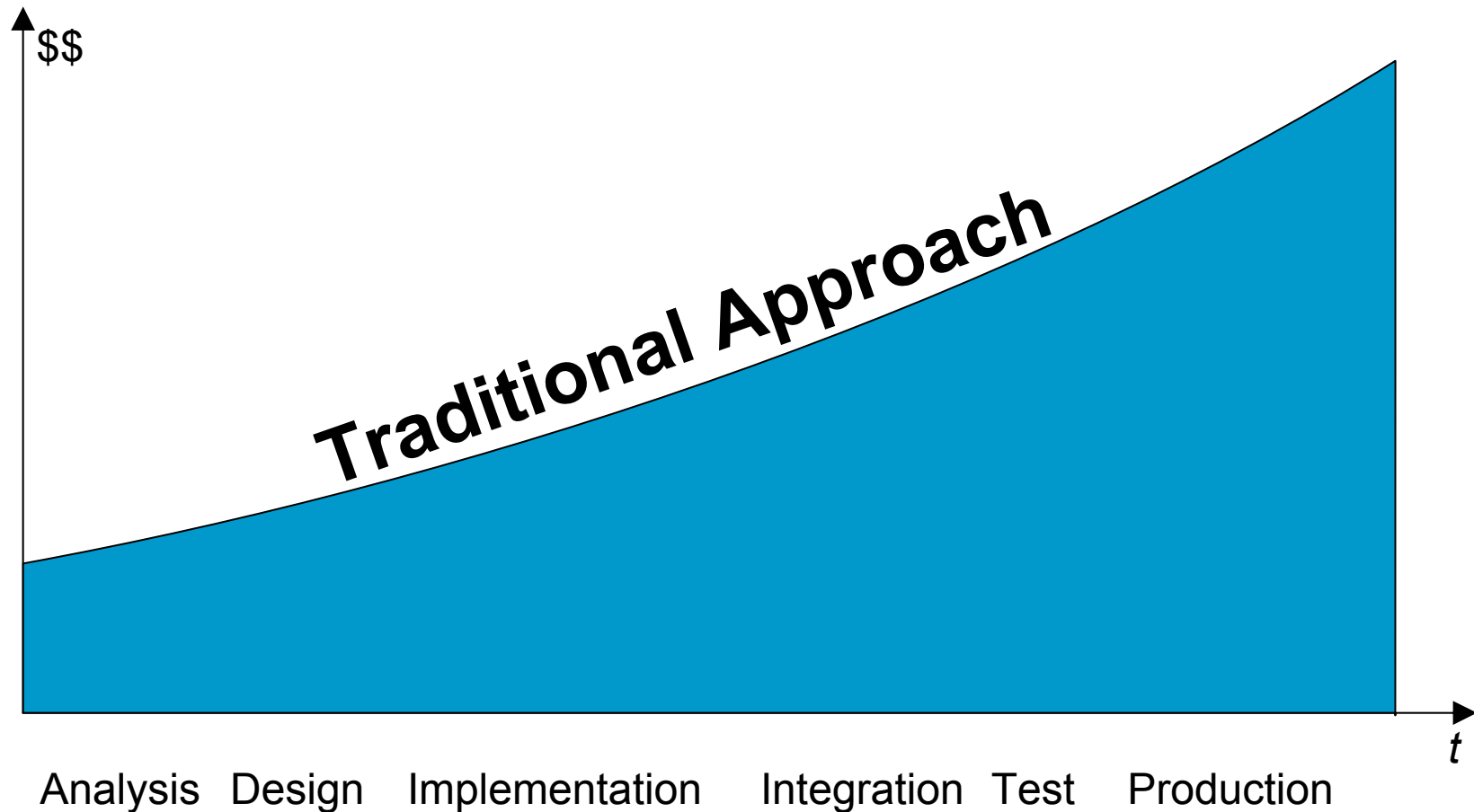
## Introduction to Extreme Programming

11-Feb-02

(c) Copyright 2001 by Manfred Lange,  
<http://www.xpexchange.net>

4

# Costs Of Fixing A Defect



# How This Is Attacked Traditionally



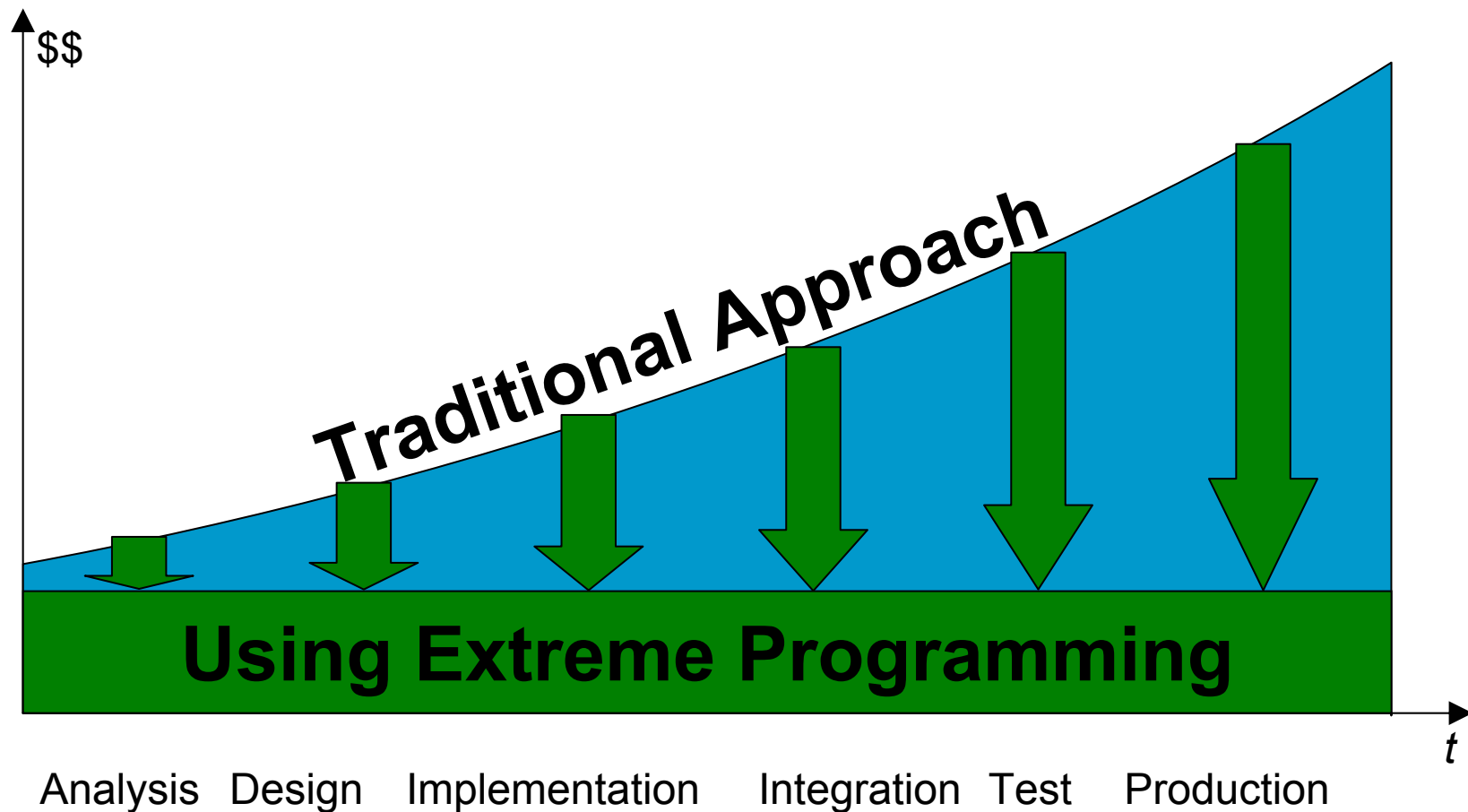
- Detect defects in early phases
- Reduce number of changes as much as possible
- How?
  - Make requirements as complete as possible
  - Make design as complete as possible
  - Make design „generic“ in order to support anticipated future needs

# How XP Adresses It



- Built change handling right into the process
- Assume that costs of change do not depend on phase or time when detected
- Make change „the default“: Embrace Change!
- Start small and work incrementally with immediate feedback

# Costs Of Fixing A Defect - Revisited



# Core Properties Of Extreme Programming



- Delivers value every 3 to 4 months
- Focusses on productive practices
  - Omits overhead, such as most paperwork
- Reduces the „truck“-factor
- Programmers work in pairs
- Does „the smallest thing that could possibly work“ (TSTTCPW)

# Values



XP is based upon 4 core values

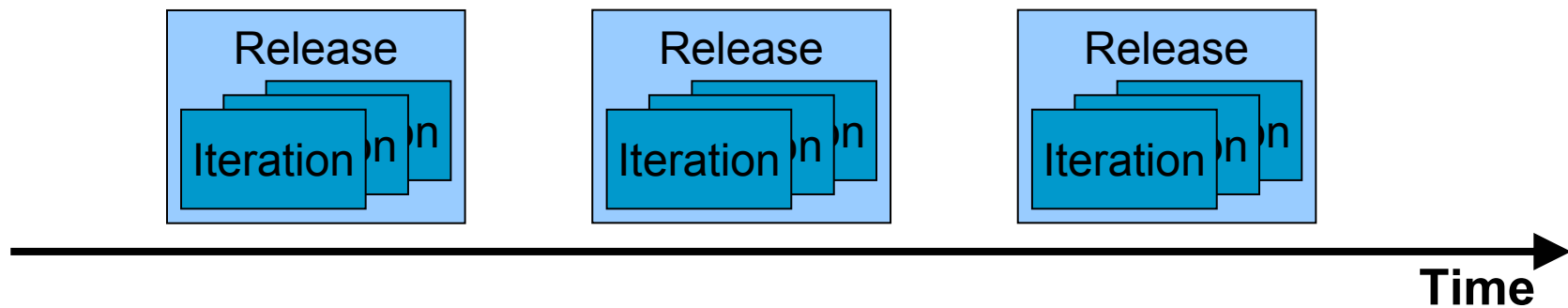
- Communication
- Simplicity
- Feedback
- Courage

# The 12 Practices



- Planning Game
- Small Releases
- Metaphor
- Simple Design
- Testing
- Refactoring
- Pair Programming
- Collective Code Ownership
- Continuous Integration
- Sustainable Pace
- Onsite Customer
- Coding Standards

# Process Timeline



# User Story Examples



- “Prevent deletion of administrator account.”
- “Display all notes for a service request, ordered by creation time.”

# Task Examples



- “Testcase: no notes for service request.”
- “Write SQL-script for creating customer table.”
- “Refactor method foo() in class xyz. Move it up the hierarchy.”

# Observations From Practice



- no dependencies between user stories
- almost no documentation yet, except
  - user story cards, task cards
- people work on project from day one
- no special tasks for interns
- high confidence in test cases
  - merciless refactoring without fear

# Heuristics



- “You aren’t gonna need it!” (YAGNI)
- “The simplest thing that could possibly work.” (TSTTCPW)
- “Once and only once.” (OAOO)
- “How little can we do, and still build great software?” (Kent Beck)

# „No More Documentation!“



False! True is:

- Paperwork is reduced as much as possible
- Documents are scheduled as „user story“, if
  - Part of the system
  - Required, e.g. for medical systems

# „XP Is For Hackers!“



False! True is:

- XP requires lots of discipline
- XP is validation centric
  - Test first programming
  - Acceptance tests accompany user stories
  - Test cases accompany tasks

# „Pair Programming Is: 2 Persons Doing The Work Of One.“



False! True is:

## ■ Studies prove:

- Pairs need 15% more time
- Quality of pairs is 15% better
- In addition:
  - Pairs exchange their knowledge
  - Pairs exchange their skills
  - Pairs exchange ideas

Source: Laurie Williams: „The Collaborative Software Process“  
Dissertation 2000, The University of Utah

# Improved Time To Market Using Pair Programming: How?

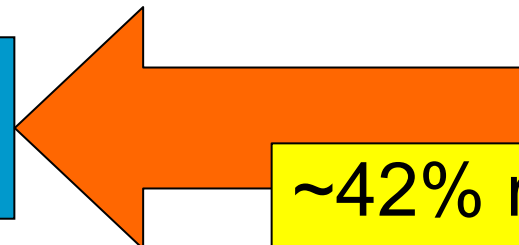
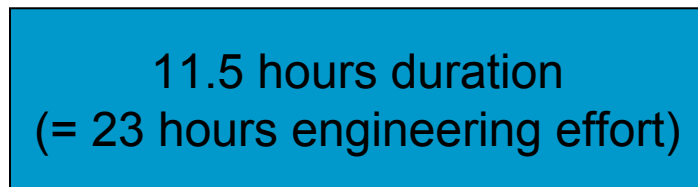


Assumption: Feature takes 20 hours effort

- No pairs



- Pairs



~42% reduced time to market!

+3 hours = +15% effort

Source: Laurie Williams: „The Collaborative Software Process“  
Dissertation 2000, The University of Utah

# Some Open Questions



- How to integrate consultants?
- How to accommodate different divergent learning styles?
- How to collaborate with non-XP team?
- How to scale up beyond 6 to 8 pairs?
- How to integrate “on-site customer” for mass product?
- When and how to adapt locally?

# How Do I Start?



- 6 essential practices
  - Planning Game
  - Small Release
  - Testing
  - Pair Programming
  - Refactoring
  - Continuous Integration
- Start, where it hurts most
- Buy an XP book and:



# Troubleshooting



- Poor quality: test cases, continuous integration
- Poor customer satisfaction: user stories, planning game, on-site customer
- Schedule not met: planning game, on-site customer
- Bad “truck factor”: pair programming
- Insufficient communication within team: pair programming, standup meeting, planning game

# Books



- Kent Beck: „Extreme Programming Explained – Embrace Change“
- Martin Fowler: „Refactoring“

# Internet



- XP Exchange

- [www.xpexchange.net](http://www.xpexchange.net)

- Don Wells

- [www.extremeprogramming.org](http://www.extremeprogramming.org)

- Ron Jeffries

- [www.xprogramming.com](http://www.xprogramming.com)



# Backup Slides

11-Feb-02

(c) Copyright 2001 by Manfred Lange,  
<http://www.xpexchange.net>

26

# A Pair Programming Desk



11-Feb-02

(c) Copyright 2001 by Manfred Lange,  
<http://www.xpexchange.net>

27

# Private Space



11-Feb-02

(c) Copyright 2001 by Manfred Lange,  
<http://www.xpexchange.net>

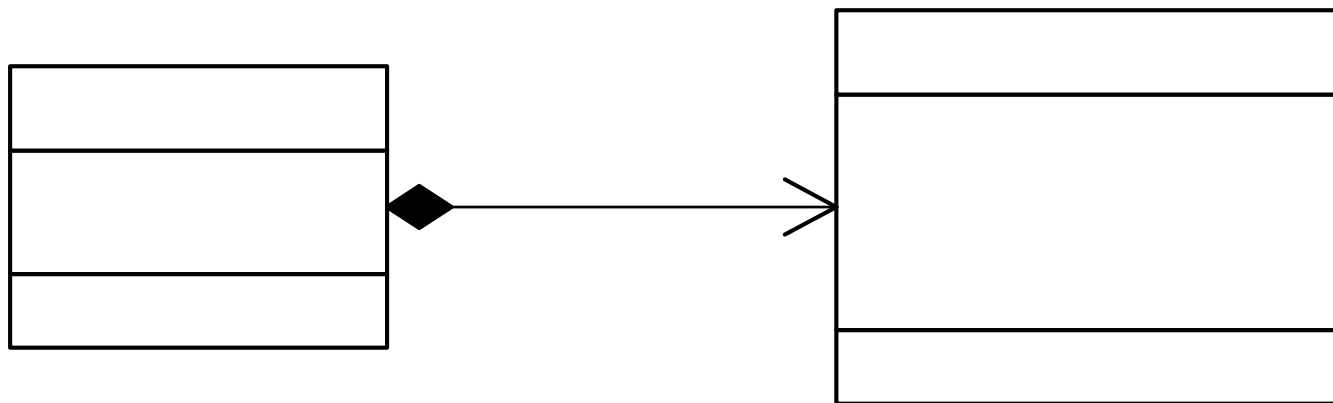
# Programming Example



- User Story: “For an invoice sum up all line items and calculated the total.”
- Sample invoice:

No	Description	Qty	Price	Line Total
1	SmartCard 3715	50	2.50	125,00
2	Reader SmartScan NT	2	75,00	150,00
	<b>Total</b>			<b>275,00</b>

# Class Diagram For Example



# Open Questions For Example



- What, if price is negative (refund)?
- What, if price is zero?
- How to handle discounts?
- What happens if there are multiple invoices with same product? Shouldn't we have a class "Product"?