Introduction to eXtreme Programming

a cura di Paolo Foletto

Java User Group Padova

Plan of presentations

- In this first presentation I will introduce the **values**, the **principles** and the **practices**.
- We will learn the terms tipical of xp and agile development.
- In the second (next) presentation will be held in March (?) and it will be more values and process oriented. The agile manifesto. Unified process versus light processes.
- Why XP works and why XP is or not accepted in Italy?

Cosa è XP?

- eXtreme Programming
- È una delle metodologie cosiddette agili per lo sviluppo di software.
- Le metodologie agili nascono per svincolarsi da metodologie troppo rigide mantenendo però un approccio **disciplinato**

Note To Programmers

Even programmers can be whole people in the real world. XP is an opportunity to test yourself, to be yourself, to realize that maybe you've been fine all along and just hanging with the wrong crowd.

Edizioni di XP

- Il libro di riferimento è
- EXtreme Programming EXplained embrace the change
- Esistono due edizioni
  - La prima del 1999
  - La seconda del 2004
- Entrambre scritte da Kent Beck e entrambe con la prefazione di Erich Gamma
  - Kent Beck e Erich Gamma hanno fatto insieme una “little thing” ....

Cosa è XP?

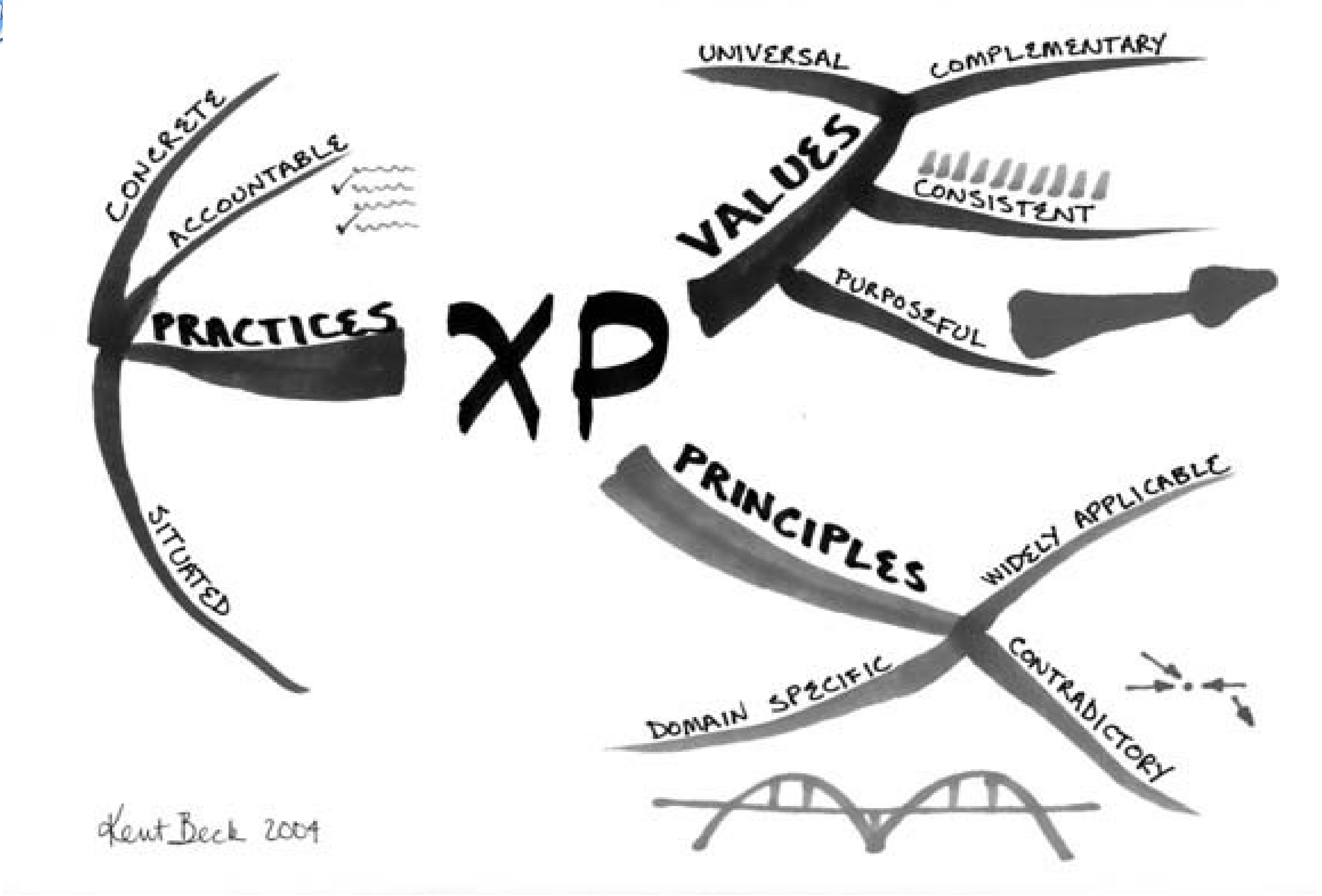
- Extreme Programming (XP) is about social change
- Incipit della seconda edizione
- Definizione data nella prima versione
- XP is a lightweight methodology for small-to medium-sized teams developing software in the face of vague or rapidly changing requirements

Cosa è XP?

- Dopo 5 anni di evoluzione e cambiamenti
- XP is lightweight
  - In XP fai solamente quello che è necessario per creare valore per il cliente
- XP is a methodology based on addressing constrains in software development.
- XP can work with team of any size
- XP adapts to vague or rapidly changing requirements

Da dove arriva?

- Nasce nella primavera del 1996 grazie a Kent Beck.
- Viene applicata in un progetto di gestione paghe alla Chrysler


XP
VALUES
UNIVERSAL
COMPLEMENTARY
CONSISTENT
PURPOSEFUL
PRACTICES
CONCRETE
ACCOUNTABLE
SITUATED
PRINCIPLES
WIDELY APPLICABLE
DOMAIN SPECIFIC
CONTRADICTORY
Kent Beck 2004

XP si basa su 4 (+1) valori

- Comunicazione
- Feedback
- Semplicità
- Coraggio
- Rispetto

Comunicazione

- Tra sviluppatori
- Tra sviluppatori e utenti finali
- Tra manager, sviluppatori e utenti finali

Feedback

- Rilasci frequenti
- Test di accettazione da parte dell'utente
- Azioni immediate in risposta ai risultati del test di accettazione

Semplicità

- Nella progettazione:
- – Scomposizione del progetto in unità piccole
- Nel codice realizzato
- Nella realizzazione dei test e quindi nel debug

Coraggio

- Nell’affrontare cambiamenti di requisiti.
- Nell’affrontare nuove tecnologie.
- Nel migliorare continuamente il codice.

Rispetto

- The previous four values point to one that lies below the surface of the other four: respect. If members of a team don't care about each other and what they are doing, XP won't work. If members of a team don't care about a project, nothing can save it.
- Every person whose life is touched by software development has equal value as a human being. No one is intrinsically worth more than anyone else. For software development to simultaneously improve in humanity and productivity, the contributions of each person on the team need to be respected. I am important and so are you.

- Values are too abstract to directly guide behavior.
- Long documents are intended to communicate, so are daily conversations. Which is the most effective? The answer depends partly on context and partly on intellectual principles. In this case, the **principle of humanity** suggests conversation meets the basic human need for connection and so is the preferred form of communication, all other things being equal. **Written communication is inherently more wasteful**. While written communication allows you to reach a large audience, it is a one-way communication. Conversation allows for clarification, immediate feedback, brainstorming together, and other things you can't do with a document. Written communication tends to be taken as fact or rejected outright, neither of which is an invitation to increased communication.

Principi

- **Humanity** People develop software
- **Economics** Somebody has to pay for all this
- **Mutual Benefit** Every activity should benefit all concerned.
- **Self-Similarity** try copying the structure of one solution into a new context, even at different scales
- **Improvement** In software development, "perfect" is a verb, not an adjective.
- **Diversity** Software development teams where everyone is alike, while comfortable, are not effective
- **Reflection** Good teams don't just do their work, they think about how they are working and why they are working.
- **Flow** Flow in software development is delivering a steady flow of valuable software by engaging in all the activities of development simultaneously
- **Opportunity** Learn to see problems as opportunities for change
- **Redundancy** Yes, redundancy
- **Failure** If you're having trouble succeeding, fail.
- **Quality** Sacrificing quality is not effective as a means of control
- **Baby Steps** It's always tempting to make big changes in big steps
- **Accepted Responsibility** Responsibility cannot be assigned; it can only be accepted

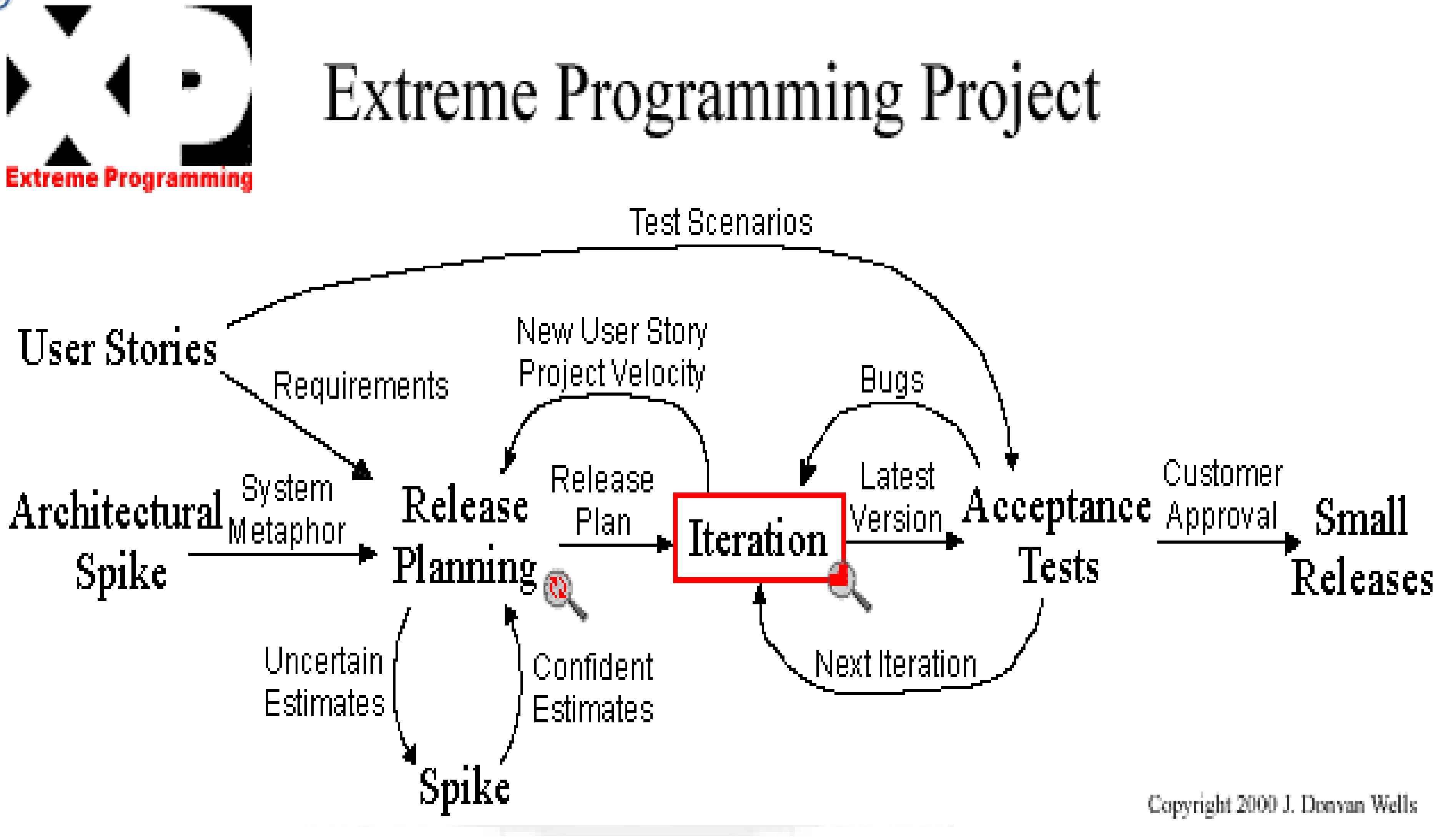
Humanity

- People develop software. This simple, inescapable fact invalidates most of the available methodological advice. Often, software development doesn't meet human needs, acknowledge human frailty, and leverage human strength.

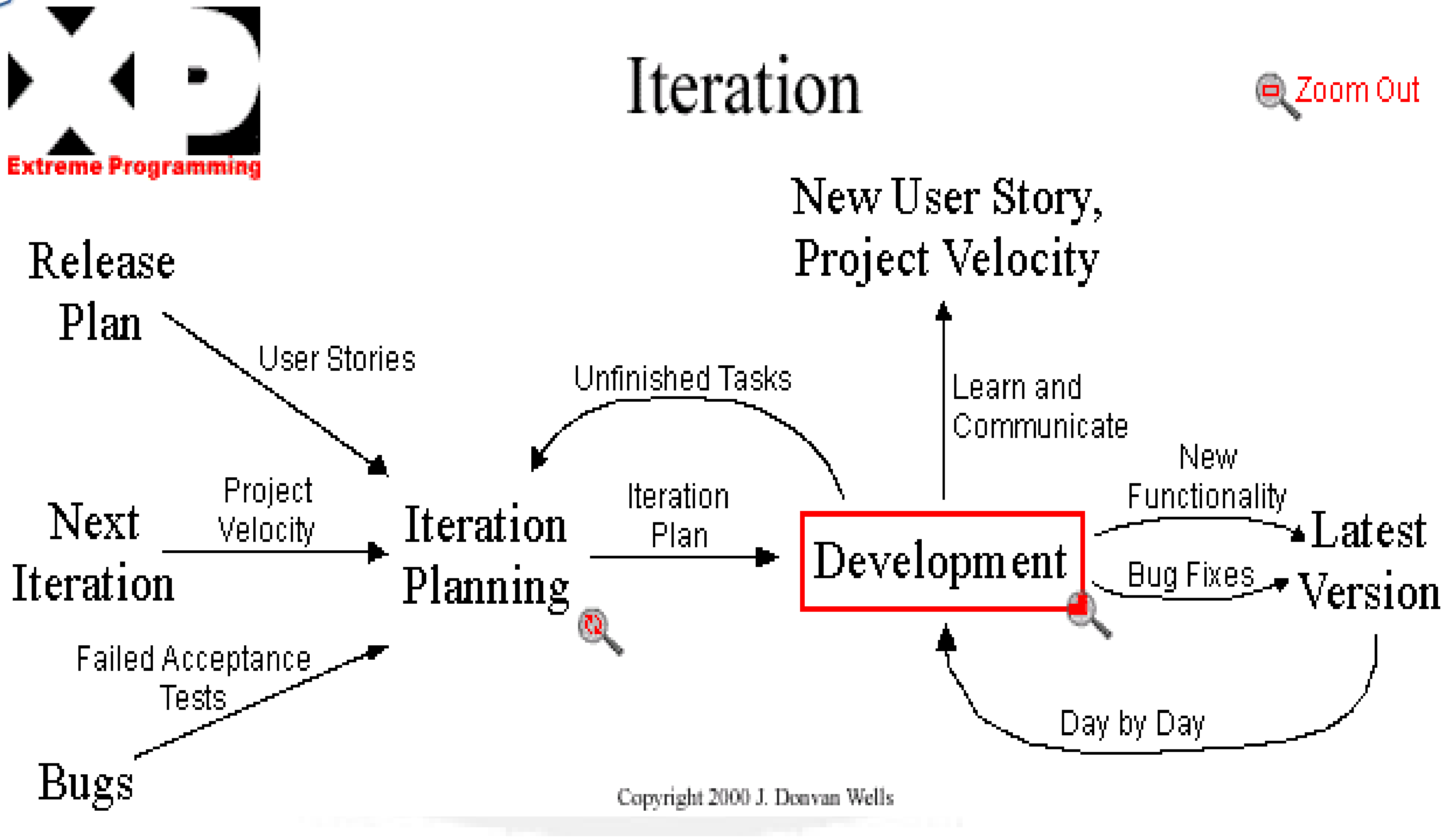
What do people need to be good developers?

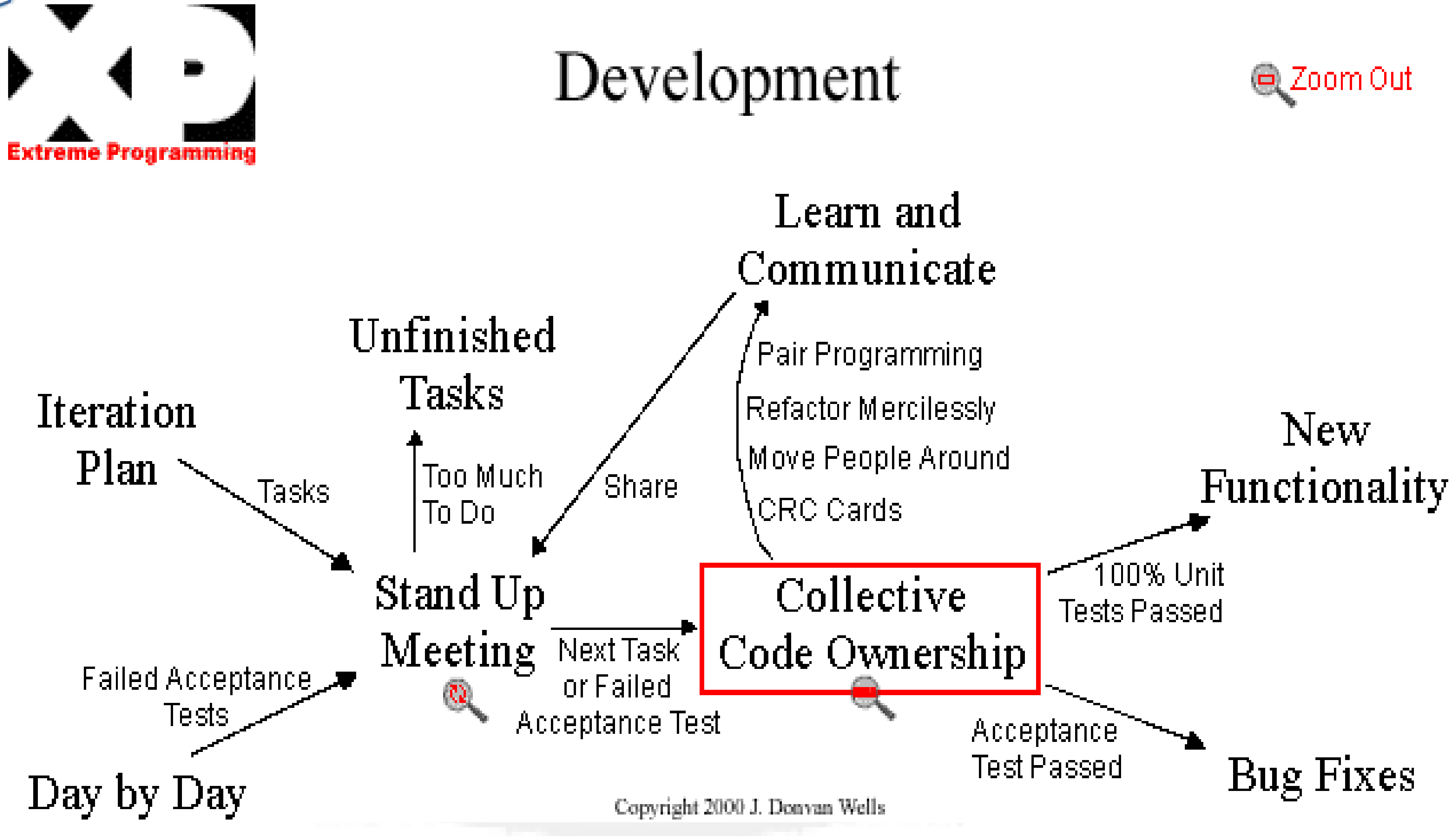
- Basic safety freedom from hunger, physical harm, and threats to loved ones. Fear of job loss threatens this need.
- Accomplishment the opportunity and ability to contribute to their society.
- Belonging the ability to identify with a group from which they receive validation and accountability and contribute to its shared goals.
- Growth the opportunity to expand their skills and perspective.
- Intimacy the ability to understand and be understood deeply by others.

Un progetto XP

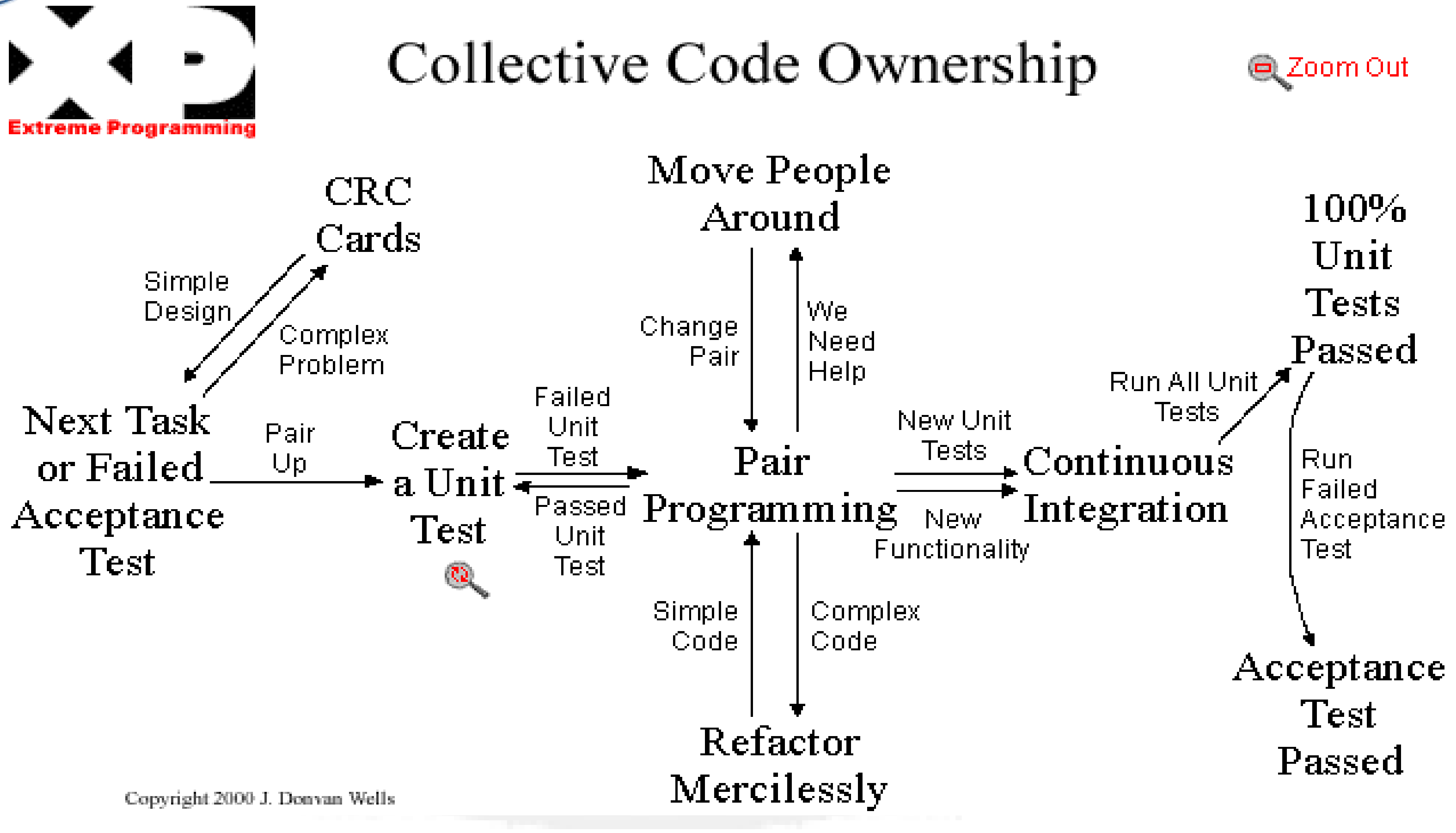


Iterazione

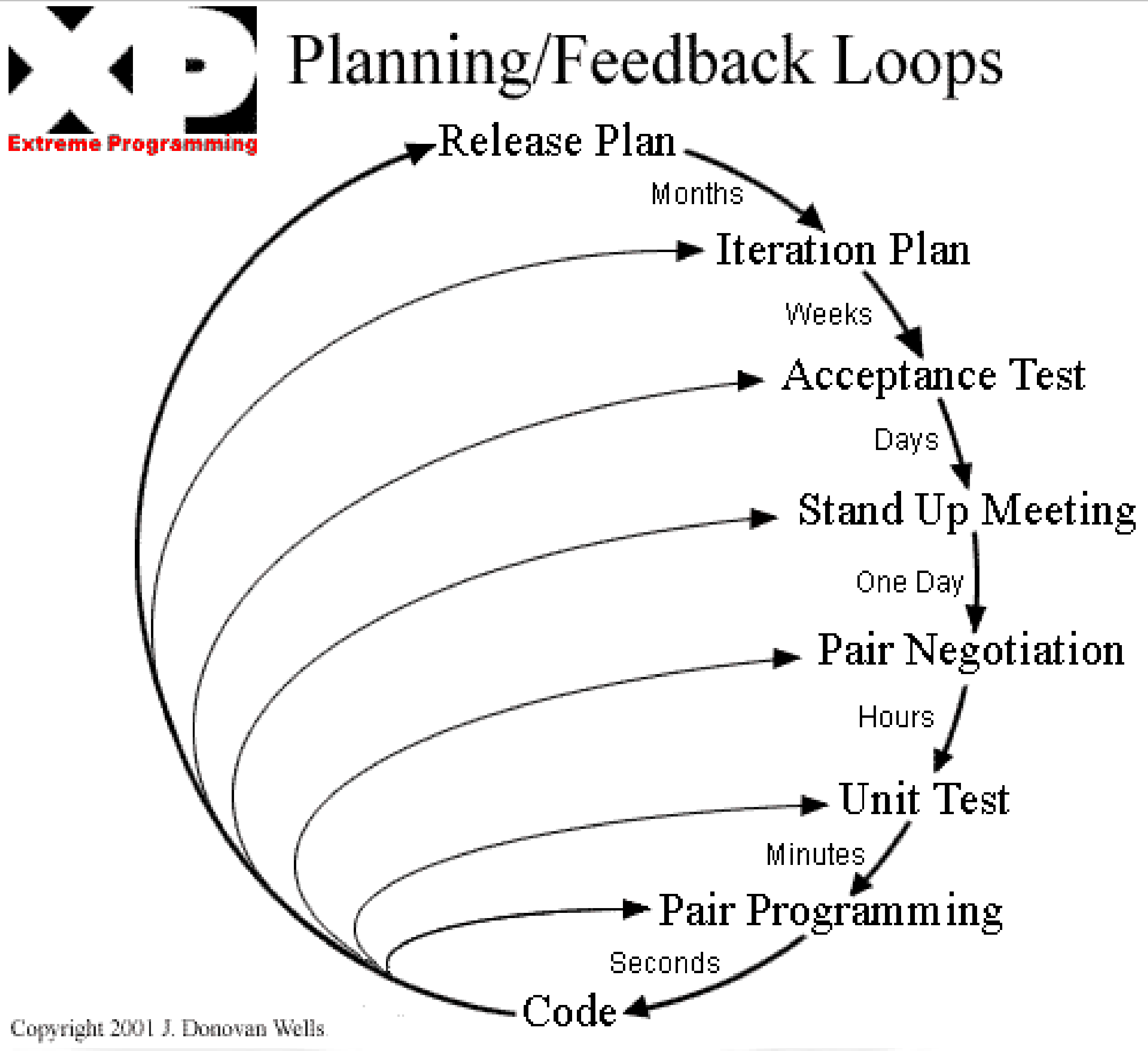

Extreme Programming
Iteration
Zoom Out
Release Plan
User Stories
Next Iteration
Project Velocity
Iteration Planning
Failed Acceptance Tests
Bugs
Unfinished Tasks
Iteration Plan
Development
New User Story, Project Velocity
Learn and Communicate
New Functionality
Bug Fixes
Latest Version
Day by Day
Copyright 2000 J. Donvan Wells


Extreme Programming
Development
Zoom Out
Learn and Communicate
Unfinished Tasks
Iteration Plan
Tasks
Too Much To Do
Share
Pair Programming
Refactor Mercilessly
Move People Around
CRC Cards
New Functionality
Stand Up Meeting
Collective Code Ownership
100% Unit Tests Passed
Next Task or Failed Acceptance Test
Failed Acceptance Tests
Day by Day
Acceptance Test Passed
Bug Fixes
Copyright 2000 J. Donvan Wells

Condivisione (?) Proprietà collettiva del codice

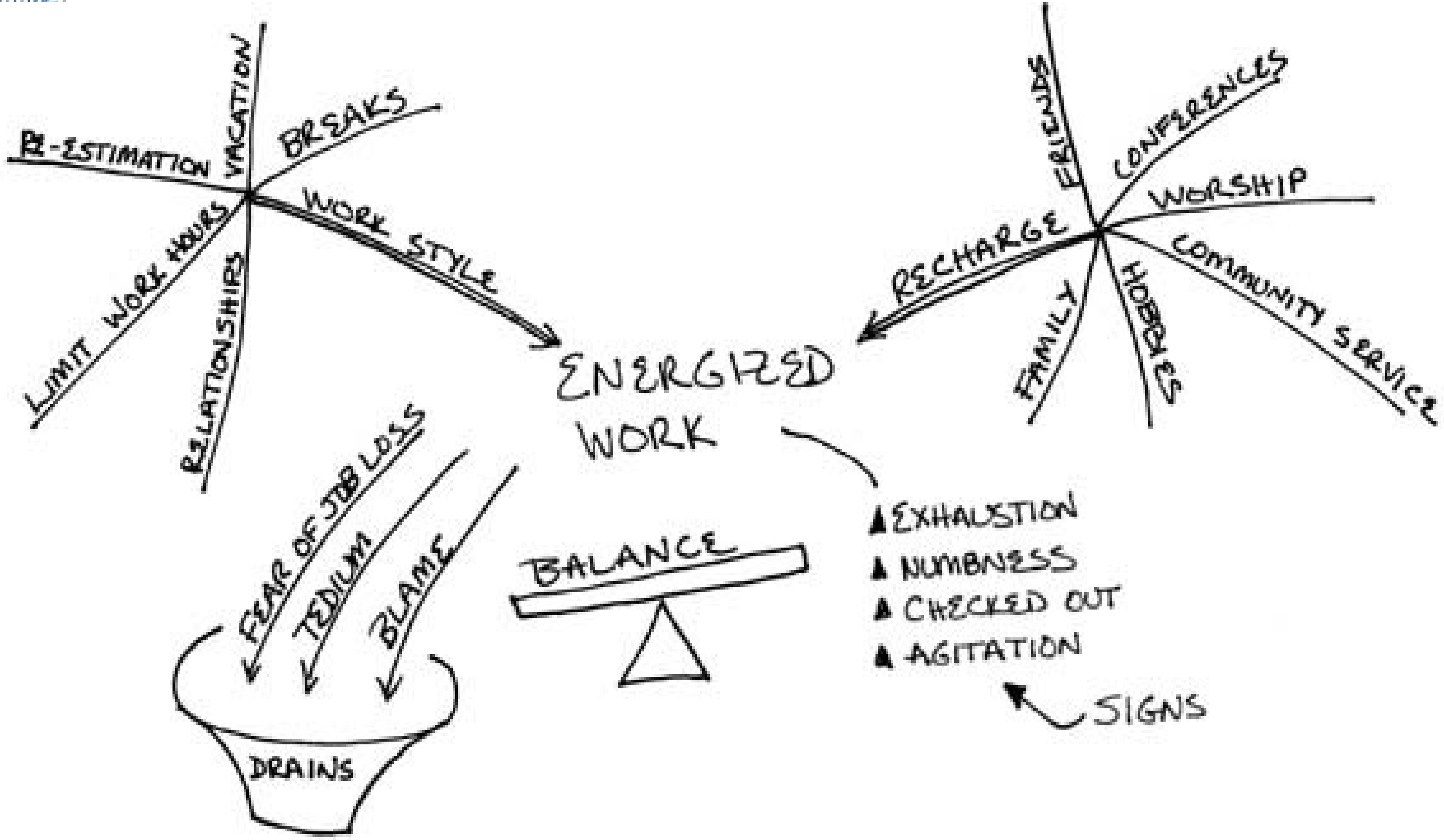


Planning/Feedback Loops



- This is the paradigm for XP. Stay aware. Adapt. Change.
- Everything in software changes. The requirements change. The design changes. The business changes. The technology changes. The team changes. The team members change. The problem isn't change, because change is going to happen; the problem, rather, is our inability to cope with change.

Energized work



- Ciascun team adatta alla propria specifica situazione ed esigenze ciascun principio

Regole e pratiche di XP - Pianificazione

- "User stories"
- – Scritte dal cliente
- – Massimo di 3 settimane di lavoro
- "Release Planning"
- – Combina esigenze di manager e clienti con le esigenze di sviluppo
- "Small releases"
- – È più facile intervenire in caso di problemi
- "Iteration planning"
- – Suddivisione in "tasks" delle "user stories"

Pratiche di XP - Codifica

Progettare il test prima

– Facilita la scrittura del codice

– "Costringe" a rispettare i requisiti definiti dalle "user stories".

"Pair Programming"

– Migliora la qualità del software.

Pratiche di XP - Codifica

“Collective Code Ownership”

- Chiunque può modificare o correggere qualsiasi modulo software.
- Responsabilità più distribuite.

“No overtime”

“Forty hours a week”

“Sustainable pace” ritmo sostenibile

- Riduce il numero di errori.
- Migliora la soddisfazione degli sviluppatori.
- Neanche aggiungere persone ad un progetto migliora situazioni critiche.

Pratiche di XP - Progettazione

"Customer (always ?) on site"

-Per definire le "user stories", per concordare i rilasci, per provare e collaudare il software il prima possibile.

"Simplicity"

– Contenere sempre le singole unità di lavoro entro certi limiti aiuta la pianificazione, il test e lo sviluppo.

"Spike solutions"

– Quando ci sono incertezze nelle stime dovute ad incognite tecnologiche, sviluppare piccoli prototipi con l'obiettivo di ridurre il rischio.

"Refactor"

– Non appena possibile modificare parti di codice complesso per semplificare

“TDD Test Driven Development”

sviluppo guidato dal test

W Bruno Bossola

“Unit tests”

- Il test deve essere creato prima della stesura del codice tramite appropriati strumenti per l’automazione che fanno parte degli strumenti di sviluppo.
- Devono essere trattati alla stessa stregua del codice.
- I moduli software modificati successivamente devono superare i test scritti precedentemente.
- Alla scoperta di un baco la prima azione da intraprendere è la riscrittura del test.

“Acceptance tests”

- Verificano che le “user stories” siano rispettate
- Vanno scritti con i clienti subito dopo aver sviluppato le “user stories”
- Solo il superamento di “acceptance tests” modifica lo stato di avanzamento del progetto.

Primary Practices

- **Sit Together** Develop in an open space big enough for the whole team
- **Whole Team** Include on the team people with all the skills and perspectives necessary for the project to succeed
- **Informative Workspace** Make your workspace about your work
- **Energized WorkWork** only as many hours as you can be productive and only as many hours as you can sustain.
- **Pair Programming** Write all production programs with two people sitting at one machine
- **Stories** Plan using units of customer-visible functionality.
- **Weekly Cycle** Plan work a week at a time
- **Quarterly Cycle** Plan work a quarter at a time
- **Slack** In any plan, include some minor tasks that can be dropped if you get behind
- **Ten-Minute Build** Automatically build the whole system and run all of the tests in ten minutes
- **Continuous Integration** Integrate and test changes after no more than a couple of hours
- **Test-First Programming** Write a failing automated test before changing any code
- **Incremental Design** Invest in the design of the system every day

Primary and corollary

- Cosa distingue le pratiche primarie da quelle corollarie?
- E' un orientamento alla diminuzione del rischio
- The practices in this chapter seem to me to be difficult or dangerous to implement before completing the preliminary work of the primary practices. If you begin deploying daily, for example, without getting the defect rate down close to zero (with pair programming, continuous integration, and test-first programming); you will have a disaster on your hands. Trust your nose about what you need to improve next. If one of the following practices seems appropriate, give it a try. It might work or you might discover that you have more work to do before you can use it to improve your development process.

Corollary practices

- **Real Customer Involvement** Make people whose lives and business are affected by your system part of the team
- **Incremental Deployment** When replacing a legacy system, gradually take over its workload beginning very early in the project
- **Team Continuity** Keep effective teams together
- **Shrinking Teams** As a team grows in capability, keep its workload constant but gradually reduce its size
- **Root-Cause Analysis** Every time a defect is found after development, eliminate the defect and its cause
- **Shared Code** Anyone on the team can improve any part of the system at any time
- **Code and Tests** Maintain only the code and the tests as permanent artifacts. Generate other documents from the code and tests
- **Single Code Base** There is only one code stream. You can develop in a temporary branch, but never let it live longer than a few hours
- **Daily Deployment** Put new software into production every night
- **Negotiated Scope Contract** Write contracts for software development that fix time, costs, and quality but call for an ongoing negotiation of the precise scope of the system
- **Pay-Per-Use** With pay-per-use systems, you charge for every time the system is used

Shared Code

- **Differenza tra la prima edizione e la seconda si passa dal "dover" al "poter" migliorare il codice**
- Anyone on the team can improve any part of the system at any time. If something is wrong with the system and fixing it is not out of scope for what I'm doing right now, I should go ahead and fix it.
- One objection I've heard is that if no one person is responsible for a piece of code, then everyone will act irresponsibly. They will make expedient changes, leaving a mess for the next person who has to touch the code. The risk of this happening is why I've listed Shared Code as a corollary practice. Until the team has developed a sense of collective responsibility, no one is responsible and quality will deteriorate. People will make changes without regard for the team-wide consequences.
- There are other models of teamwork besides "every man for himself." The team members can collectively assume responsibility not just for the quality of what they deliver to users but also for the pride they take in their work along the way. Pair programming helps teammates demonstrate their commitment to quality to each other and helps them normalize their expectations for what constitutes quality.
-

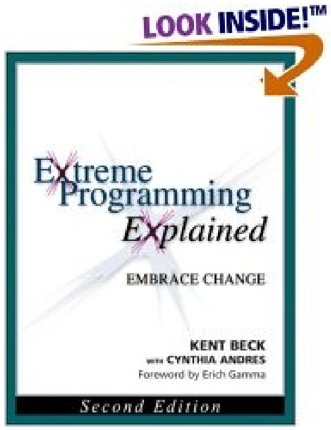
The whole XP Team

- **Testers** Testers on an XP team help customers choose and write automated system-level tests in advance of implementation and coach programmers on testing techniques
- **Interaction Designers** Interaction designers on an XP team choose overall metaphors for the system, write stories, and evaluate usage of the deployed system to find opportunities for new stories
- **Architects** Architects on an XP team look for and execute large-scale refactorings, write system-level tests that stress the architecture, and implement stories
- **Project Managers** Project managers on an XP team facilitate communication inside the team and coordinate communication with customers, suppliers, and the rest of the organization
- **Product Managers** In XP, product managers write stories, pick themes and stories in the quarterly cycle, pick stories in the weekly cycle, and answer questions as implementation uncovers under-specified areas of stories
- **Executives** Executives provide an XP team with courage, confidence, and accountability

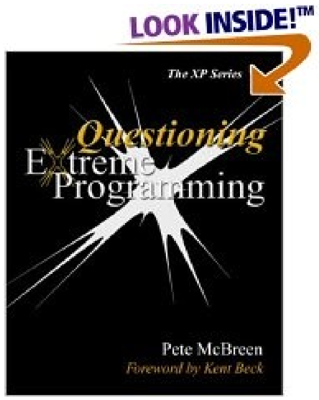
- **Technical Writers** The role of technical publications on an XP team is to provide early feedback about features and to create closer relationships with users
- **Users** Users on an XP team help write and pick stories and make domain decisions during development
- **Programmers** Programmers on an XP team estimate stories and tasks, break stories into tasks, write tests, write code to implement features, automate tedious development process, and gradually improve the design of the system. Programmers work in close technical collaboration with each other, pairing on production code, so they need to develop good social and relationship skills.
- **Human Resources** Two challenges have been reported for human resources when teams begin applying XP: reviews and hiring. The problem with reviews is that most reviews and raises are based on individual goals and achievements, but XP focuses on team performance. If a programmer spends half of his time pairing with others, how can you evaluate his individual performance? How much incentive does he have to help others if he will be evaluated on individual performance?
- Evaluating XP team members individually need not be much different from evaluating them before applying XP. In XP, valuable employees:
  - Act respectful. Play well with others. Take initiative. Deliver on their commitments.
-

Bibliografia

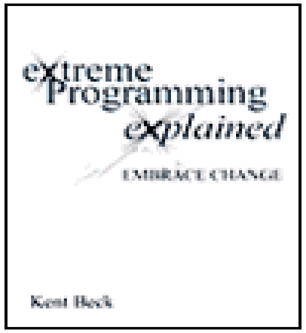
- Extreme Programming Explained: Embrace Change (2nd Edition) by Kent Beck



- Questioning Extreme Programming



- Extreme Programming Explained: Embrace Change (1st edition) by Kent Beck



Alcuni riferimenti in italia

- **http://www.xplabs.it/**
  - La prima società in italia
- **http://www.siforge.org/articles/2003/03/09-arrivare-a-xp.html**
  - Arrivare a XP (eXtreme Programming) - Intervista a Francesco Cirillo
- **http://digilander.libero.it/bbossola/seminari.html**
  - I seminari di Bruno Bossola
- http://milano-xpug.pbwiki.com/

Bibliografia

- http://www.objectmentor.com/
- http://www.extremeprogramming.org/
- http://www.xprogramming.com
- http://www.junit.org/index.htm
- http://threeriversinstitute.org
- http://c2.com/cgi/wiki e in particolare
- http://c2.com/cgi/wiki?PeopleProjectsAndPatterns
- http://www.extremejava.com/

Informazioni sul JUG Padova

- Sito Web:
  - `http://www.jugpadova.it`
- Mailing List:
  - !Attenzione nuova mailing list su googlegroups
  - `http://groups.google.com/group/jugpadova`
- Persone di riferimento
  - Dario Santamaria (dario.santamaria@jugpadova.it)
  - Lucio Benfante (lucio.benfante@jugpadova.it)
  - Paolo Donà (paolo.dona@jugpadova.it)