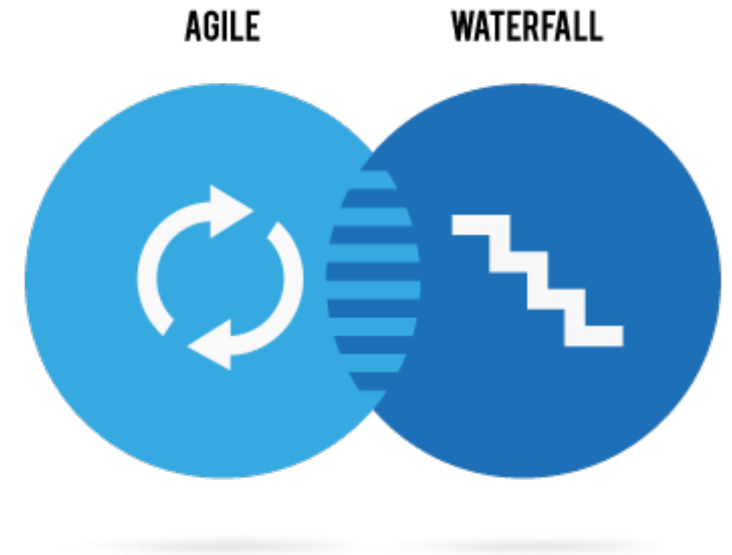
Modelli di processo per lo sviluppo del software: i modelli agili

Corso di Ingegneria del Software
CdL Informatica Università di Bologna

Obiettivi di questa lezione

- I modelli di processo agili
- Il manifesto agile
- Extreme Programming (XP)



La pianificazione spesso fallisce

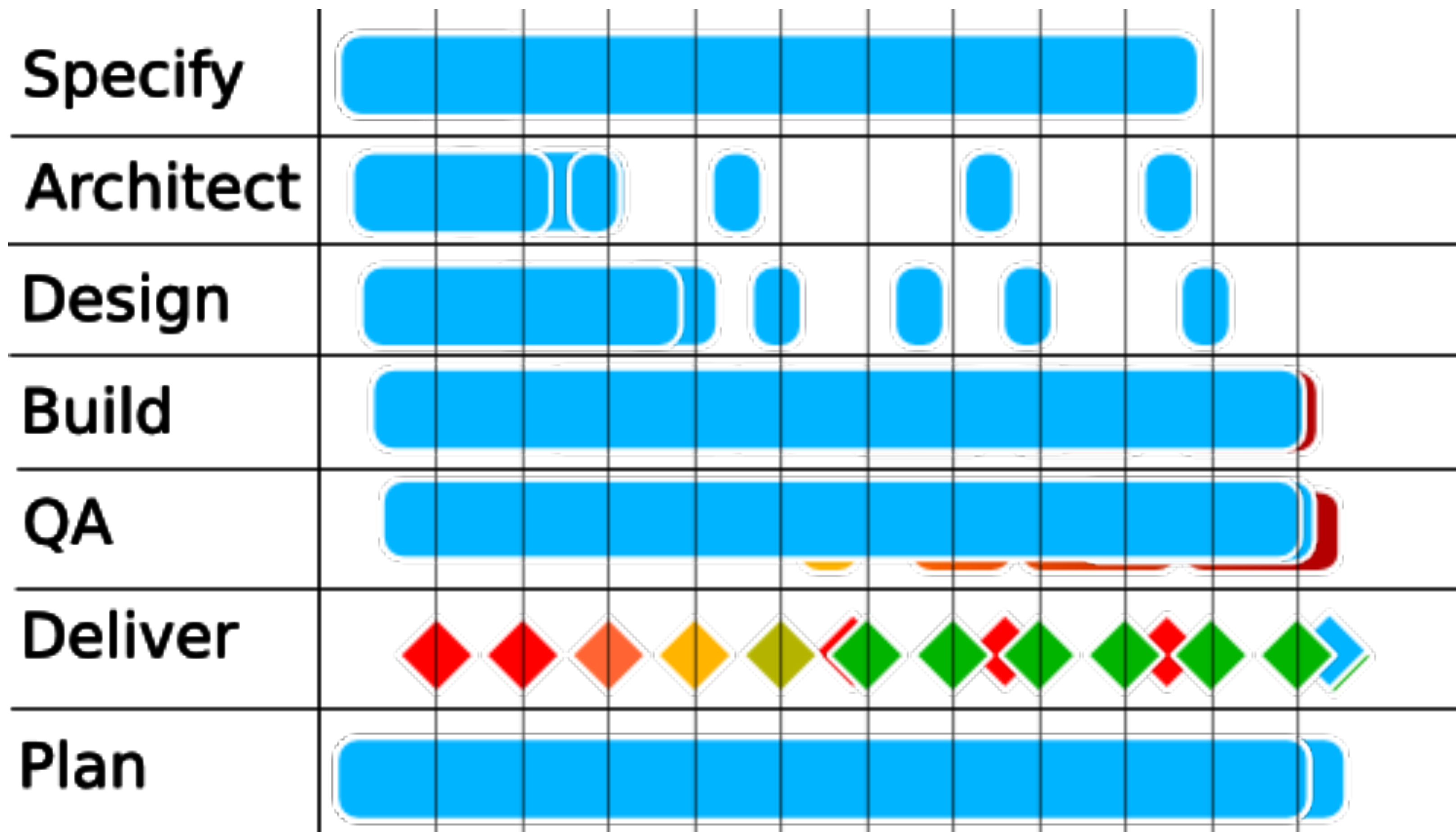
- Alcuni problemi sono complessi, richiedono il contributo di molte persone
- La soluzione all'inizio non è chiara
- I requisiti del prodotto-soluzione durante lo sviluppo probabilmente cambieranno
- Il prodotto può essere consegnato in versioni incrementali successive
- È richiesta una collaborazione stretta e un feedback rapido e continuo dagli utenti finali

Sviluppare software è un’attività sociale

- Quasi tutti i problemi difficili nello sviluppo software riguardano le persone e non le tecnologie.
- Scrivere software non è difficile.
- Tuttavia, scrivere il software *giusto* è molto difficile, perché occorre capire i bisogni delle persone, negoziare i tempi di consegna, distribuire i compiti, fare compromessi
- Le persone capaci di fare bene il software sono rare: quando hanno successo di solito non è per l’abilità tecnica di programmazione che dimostrano, ma per le loro abilità relazionali interpersonali

L’interesse dell’utente:

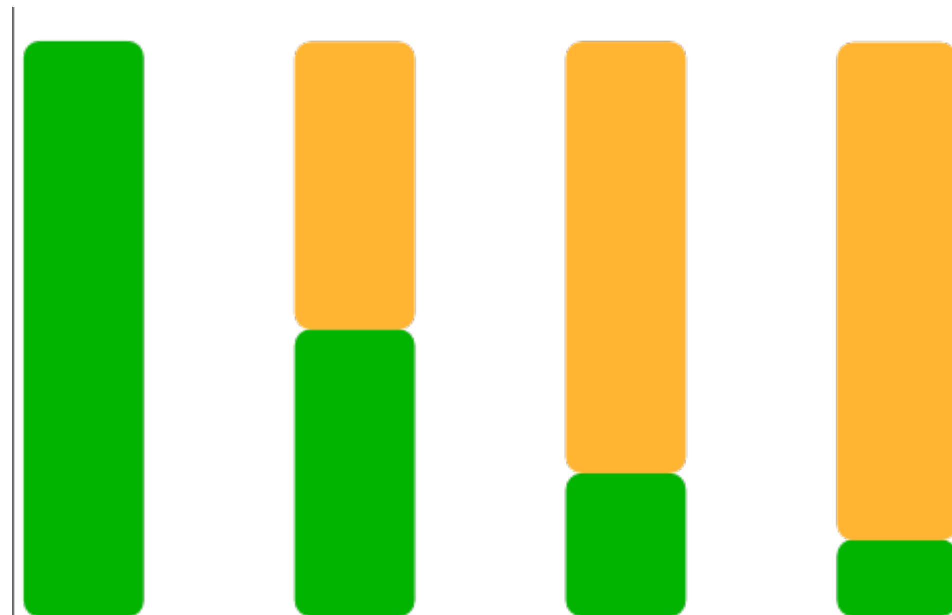
in quale riga lo vedete?



Volatilità dei requisiti

Nella slide precedente, all’utente interessa solo la riga «Deliver»: quando sarà consegnato il prodotto?

Ogni sei mesi – o anche meno – metà dei requisiti di un prodotto software perdono di interesse



www.infoq.com/articles/the-curse-of-the-change-control-mechanism

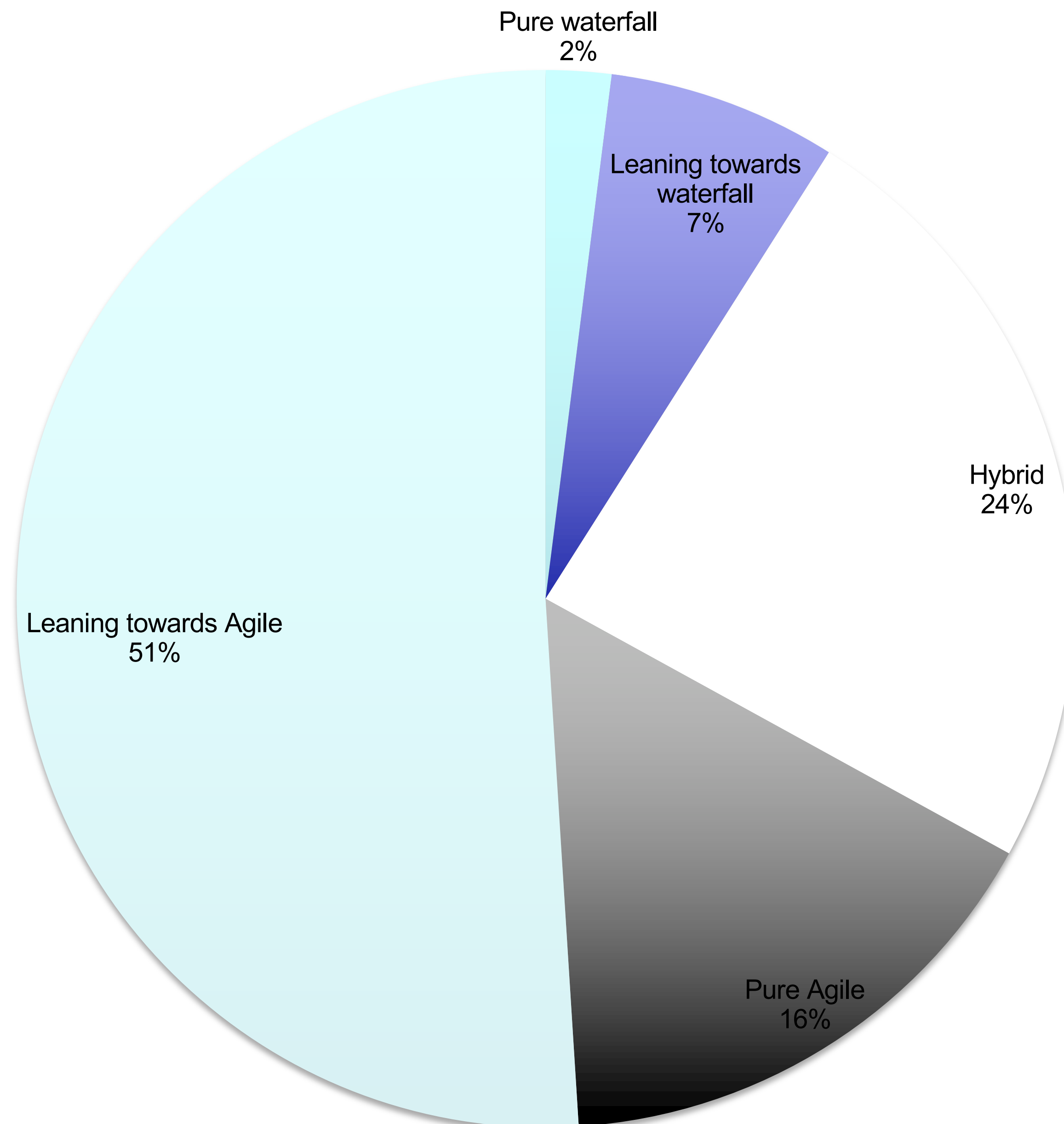
Migliorare i processi iterativi

- I processi iterativi devono produrre valore per gli stakeholder in modo incrementale e senza sprechi
- La filosofia agile ha lo scopo di evitare gli sprechi e le perdite di tempo – le attività inutili

Consegnate software di valore
Evitate gli sprechi

- www.youtube.com/watch?v=Tj-lavaMkxU

Primary development method
(source: 2015 HP survey of 601 sw professionals)



Etica del Movimento Agile

Stiamo scoprendo modi migliori di costruire il software facendolo e aiutando altri a farlo. Attribuiamo valore a:

Individui e interazioni più che a processi e strumenti
Software che funziona più che a documentazione completa
Collaborazione col cliente più che a negoziazione contrattuale
Reagire al cambiamento più che a seguire un piano

I valori a destra sono importanti, ma noi preferiamo quelli a sinistra

www.agilemanifesto.org

I principi agili

1. La nostra priorità è soddisfare il cliente mediante consegne anticipate e continue di software di valore
2. Le persone dell'azienda e gli sviluppatori devono quotidianamente lavorare assieme durante tutto il progetto
3. Le modifiche ai requisiti sono benvenute, anche nelle ultime fasi dello sviluppo
4. Consegnare di frequente software funzionante
5. Il software funzionante è la prima misura di progresso
6. I progetti si costruiscono attorno a individui motivati. Dategli l'ambiente ed il supporto di cui hanno bisogno, e confidate che facciano il loro lavoro
7. Le migliori architetture, requisiti, e design emergono da team auto-organizzanti
8. Il metodo più efficace ed efficiente di trasmettere informazioni verso e all'interno di uno sviluppo è mediante conversazioni faccia a faccia
9. I processi agili promuovono lo sviluppo sostenibile
10. L'attenzione costante all'eccellenza tecnica e al buon design esaltano l'agilità
11. La semplicità è essenziale
12. I team di sviluppo valutano la propria efficacia ad intervalli regolari e modificano di conseguenza il proprio comportamento

I principi agili

1. La nostra priorità è soddisfare il cliente mediante consegne anticipate e continue di software di valore
2. Le persone dell'azienda e gli sviluppatori devono quotidianamente lavorare assieme durante tutto il progetto
3. Le modifiche ai requisiti sono benvenute, anche nelle ultime fasi dello sviluppo
4. Consegnare di frequente software funzionante
5. Il software funzionante è la prima misura di progresso
6. I progetti si costruiscono attorno a individui motivati. Dategli l'ambiente ed il supporto di cui hanno bisogno, e confidate che facciano il loro lavoro
7. Le migliori architetture, requisiti, e design emergono da team auto-organizzanti
8. Il metodo più efficace ed efficiente di trasmettere informazioni verso e all'interno di uno sviluppo è mediante conversazioni faccia a faccia
9. I processi agili promuovono lo sviluppo sostenibile
10. L'attenzione costante all'eccellenza tecnica e al buon design esaltano l'agilità
11. La semplicità è essenziale
12. I team di sviluppo valutano la propria efficacia ad intervalli regolari e modificano di conseguenza il proprio comportamento

I principi agili

Knowledge TRAIN

The 12 agile principles*

1 Satisfy the **customer**

2 Welcome **change**

3 Deliver **frequently**

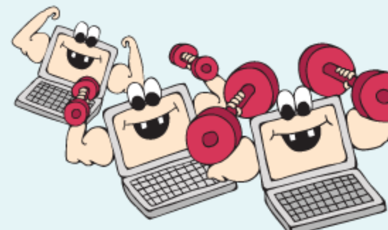
Sprint 1	Sprint 2	Sprint 3	Sprint 4	Sprint 5
story	story	story	story	story
story	story	story	story	story
story	story	story	story	story

4 Work **together**

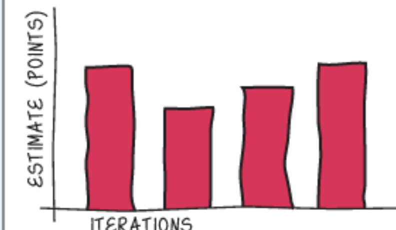
5 Trust and **support**

6 Face-to-face **conversation**

7 Working **software**



8 Sustainable **development**



9 Continuous **attention**

10 Maintain **simplicity**

11 Self-organizing **teams**

12 Reflect and **adjust**

Copyright © 2016 Knowledge Train Limited. www.knowledgetrain.co.uk. *Quoted from www.agilemanifesto.org

Metodi agili

I metodi agili sono una famiglia di metodi di sviluppo che hanno in comune:

- **Rilasci frequenti** del prodotto sviluppato in modo incrementale
- **Collaborazione** continua del team di sviluppo col **cliente**
- **Documentazione** di sviluppo **ridotta**
- **Valutazione** sistematica e continua di **valori** e **rischi** dei **cambiamenti**

Lavorare per progetti vs lavorare per prodotti

	Progetto	Prodotto
Scope (portata del progetto, cioè risultati attesi)	Funzioni decise all'inizio	Lista di funzioni "correnti" in priorità
Agenda dei rilasci	Durata fissa	Rilasci multipli
Budget	Allocato all'inizio	Ciclico

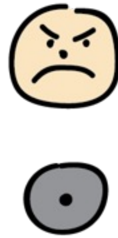
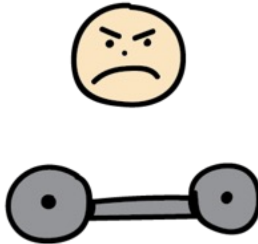
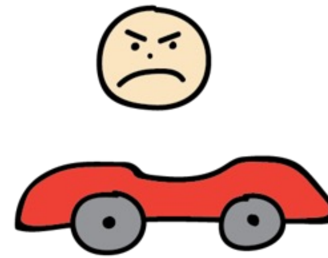
Minimal viable product MVP

- Nello sviluppo di un nuovo prodotto, il prodotto minimo funzionante (**Minimum Viable Product "**MVP") è il prodotto con il più alto ritorno sugli investimenti rispetto al rischio
- È una strategia mirata ad evitare di costruire prodotti che i clienti non vogliono, che cerca di massimizzare le informazioni apprese sul cliente per ogni euro speso.
- Un MVP non è, quindi, un prodotto minimo, ma un processo iterativo di generazione di idee, prototipazione, presentazione, raccolta dati, analisi ed apprendimento.
- MVP è un fondamento del movimento “Lean”, alla base di Agile

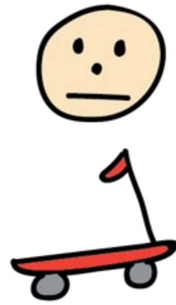
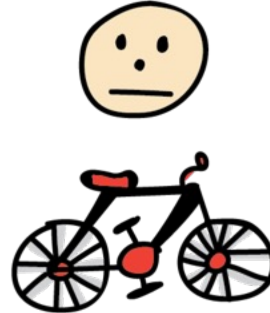
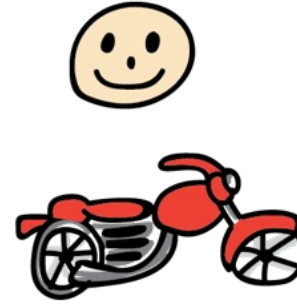
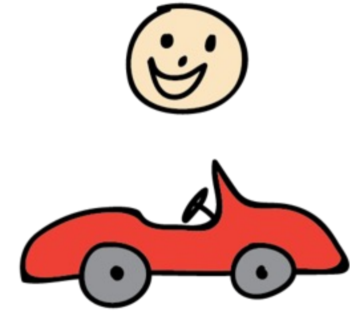
Fonte: Wikipedia

Minimal Viable Product
(prodotto minimo funzionante)

Not Like This!

Like This!

Minimum Lovable Product

Jeff Wilke

Source: Fact of the Day 1 interview with Jeff Wilke June 16, 2021
Published: January 2021

Minimum loveable product

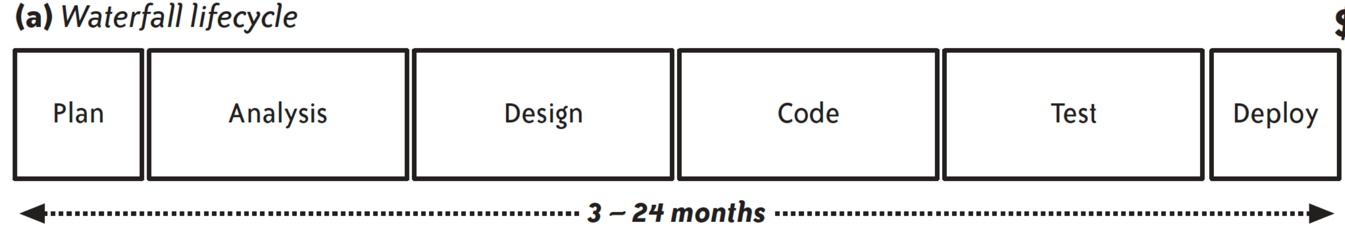
A minimum viable product (MVP) is a version of a product with just enough features to be usable by early customers who can then provide feedback for future product development.

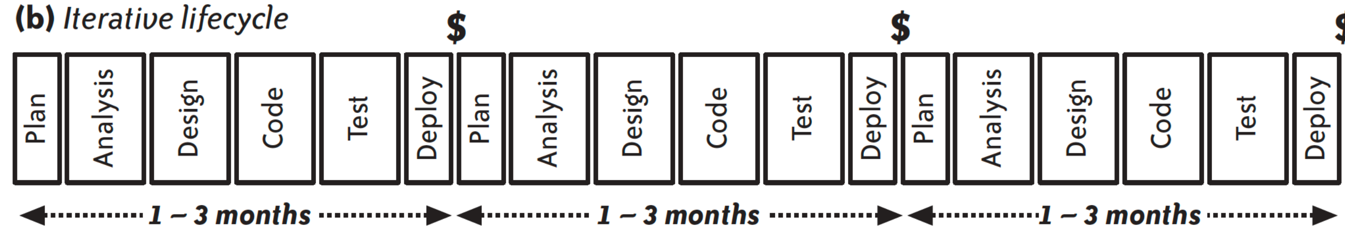
Jeff Wilke recognized that Amazon's MVPs failed to insist on high standards. Teams were making trade-offs to prioritize speed over customer delight.

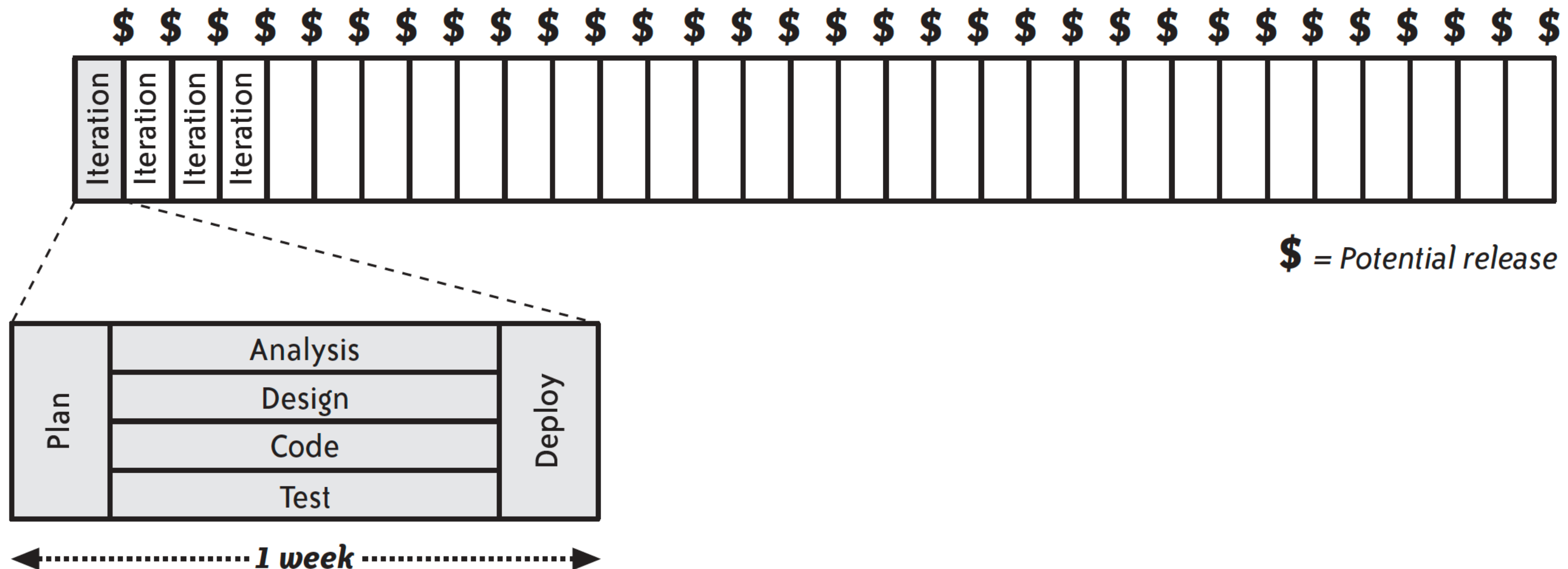
A minimum loveable product (MLP) is a term used at Amazon to describe a version of a product with just enough features for early customers to love the experience.

"It may be that some customers are ok with MVP, but customers don't want MVP. It has to be lovable or they're not going to adopt it. We had to change the standard for how we were thinking about press releases and products. We were sliding toward a standard that was too low." - Jeff Wilke

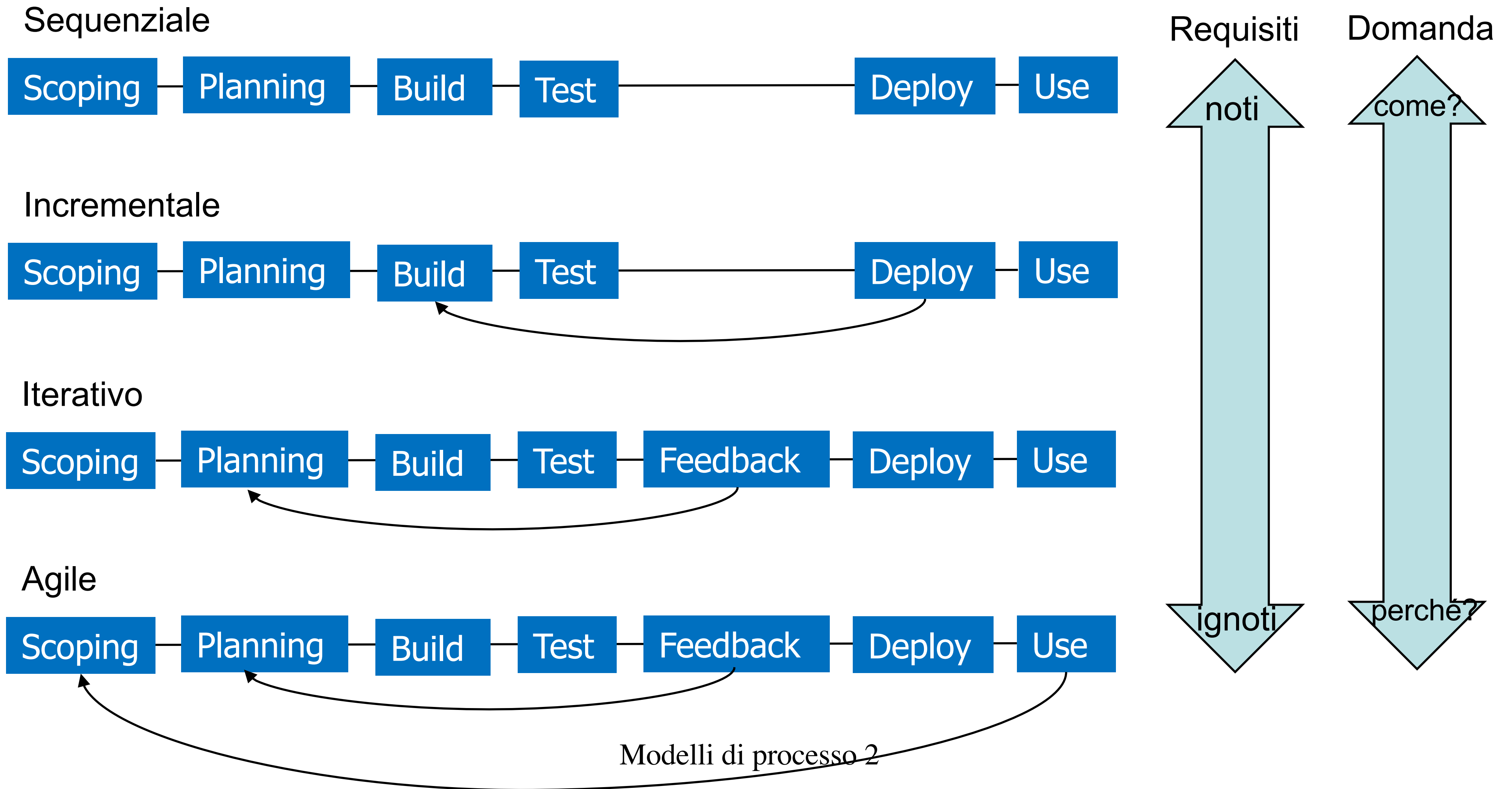
Waterfall vs iterativo vs agile







La diversità dei modelli di processo



Metodi agili

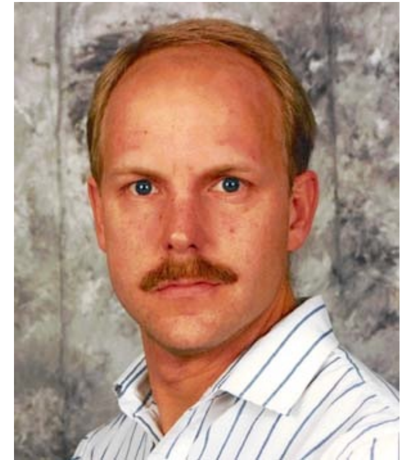
- Extreme Programming (XP)
- Scrum
- Feature-Driven Development (FDD)
- Adaptive Software Process
- Crystal Light Methodologies
- Dynamic Systems Development Method (DSDM)
- Lean Development

- DevOps
- SAFe
- Less

eXtreme Programming (XP)

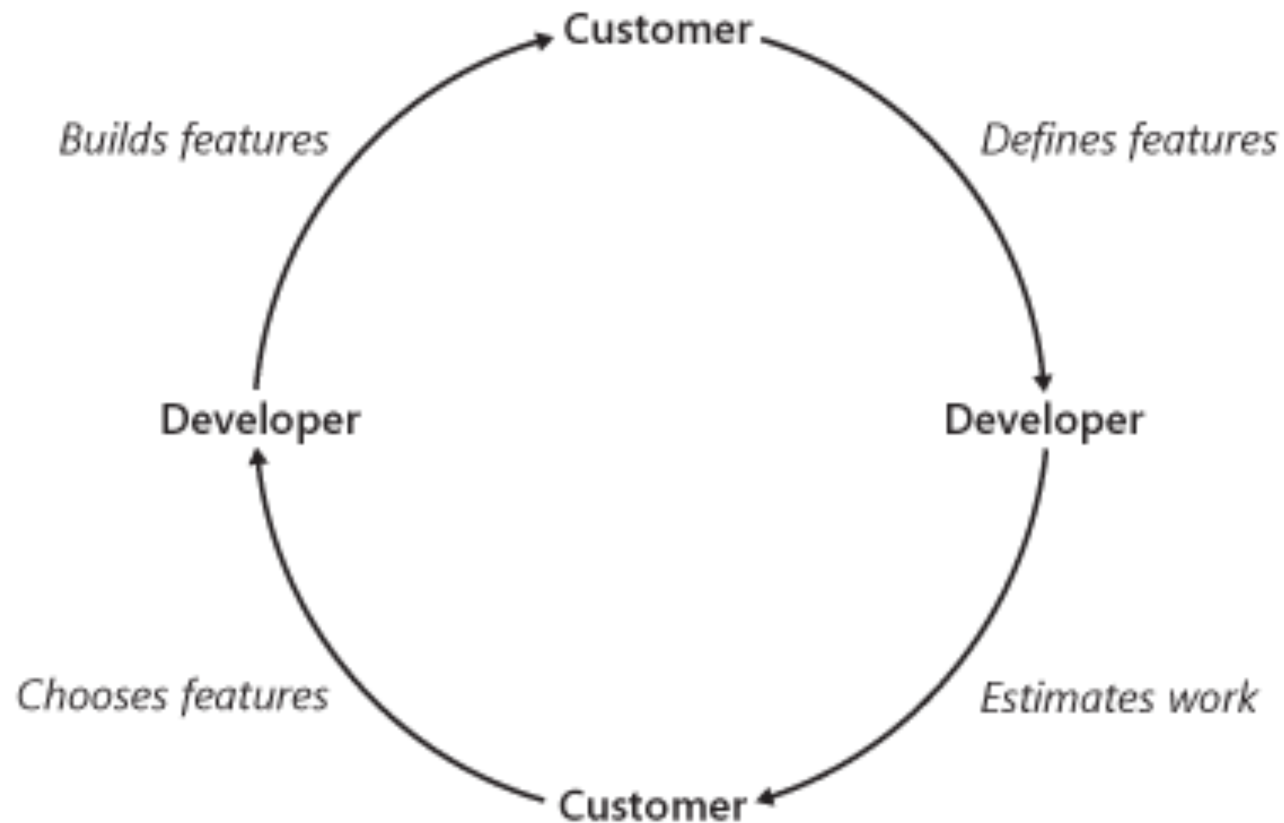
“Extreme Programming è una disciplina di sviluppo software basata sui valori di

- *semplicità*,
- *comunicazione*,
- *feedback*, e
- *coraggio*”.



Kent Beck

La base della collaborazione agile in XP



Il team XP

- Il team di sviluppo, di solito costituito da meno di dieci persone, è riunito nello stesso locale
- E’ sempre presente un rappresentante del cliente, capace di rispondere a domande del team riguardo ai requisiti
- Il team deve usare comportamenti di sviluppo **semplici**, allo stesso tempo **capaci di informare tutti** sullo stato del progetto ma anche di **adattare** i comportamenti alla situazione specifica

eXtreme Programming (XP)

www.extremeprogramming.org

Extreme Programming Project

Test Scenarios
User Stories
Requirements
New User Story
Project Velocity
Bugs
Architectural Spike
System Metaphor
Release Planning
Release Plan
Iteration
Latest Version
Acceptance Tests
Customer Approval
Small Releases
Uncertain Estimates
Confident Estimates
Next Iteration
Spike

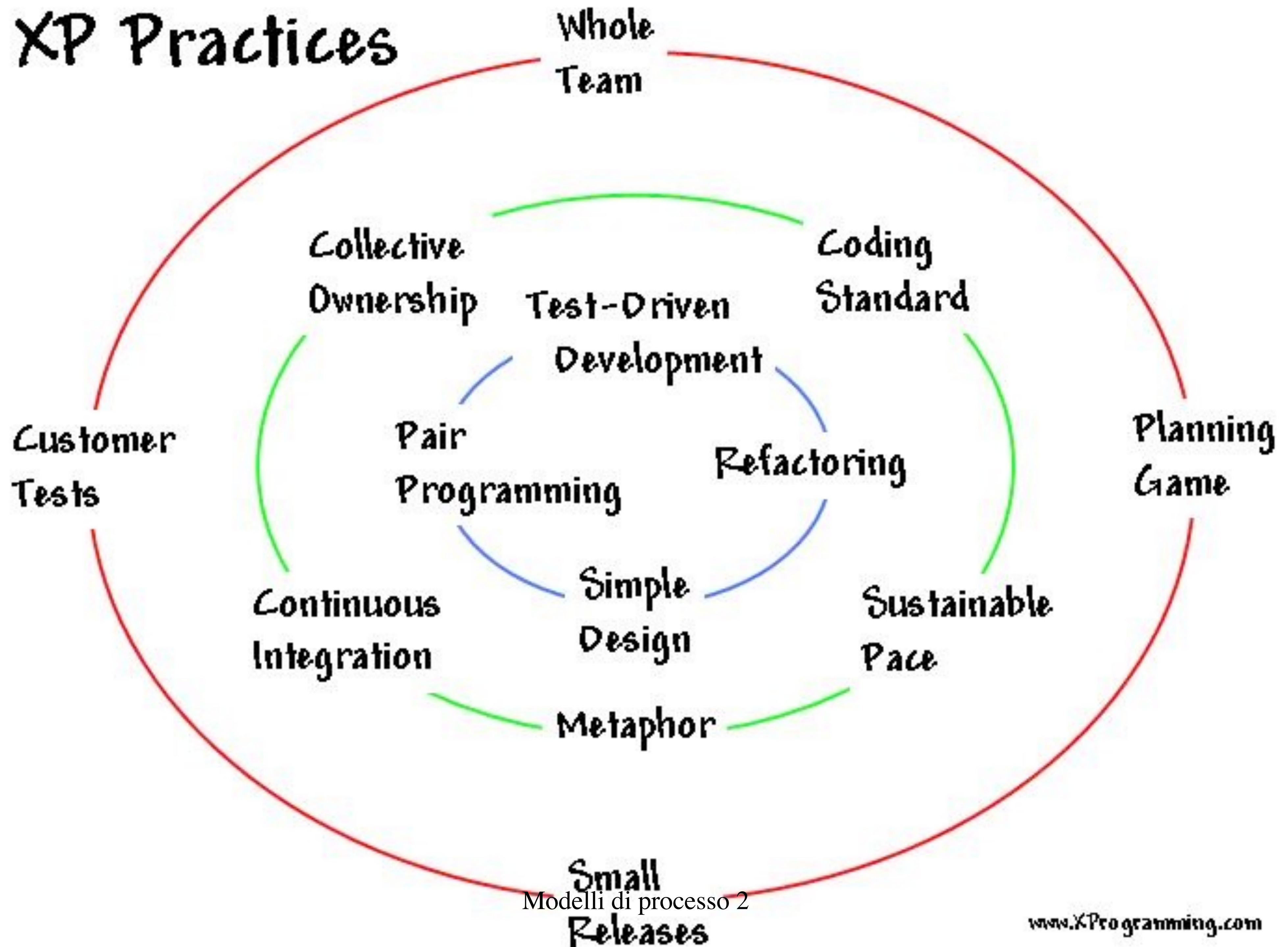
Copyright 2000 J. Donvan Wells

Exploration phase

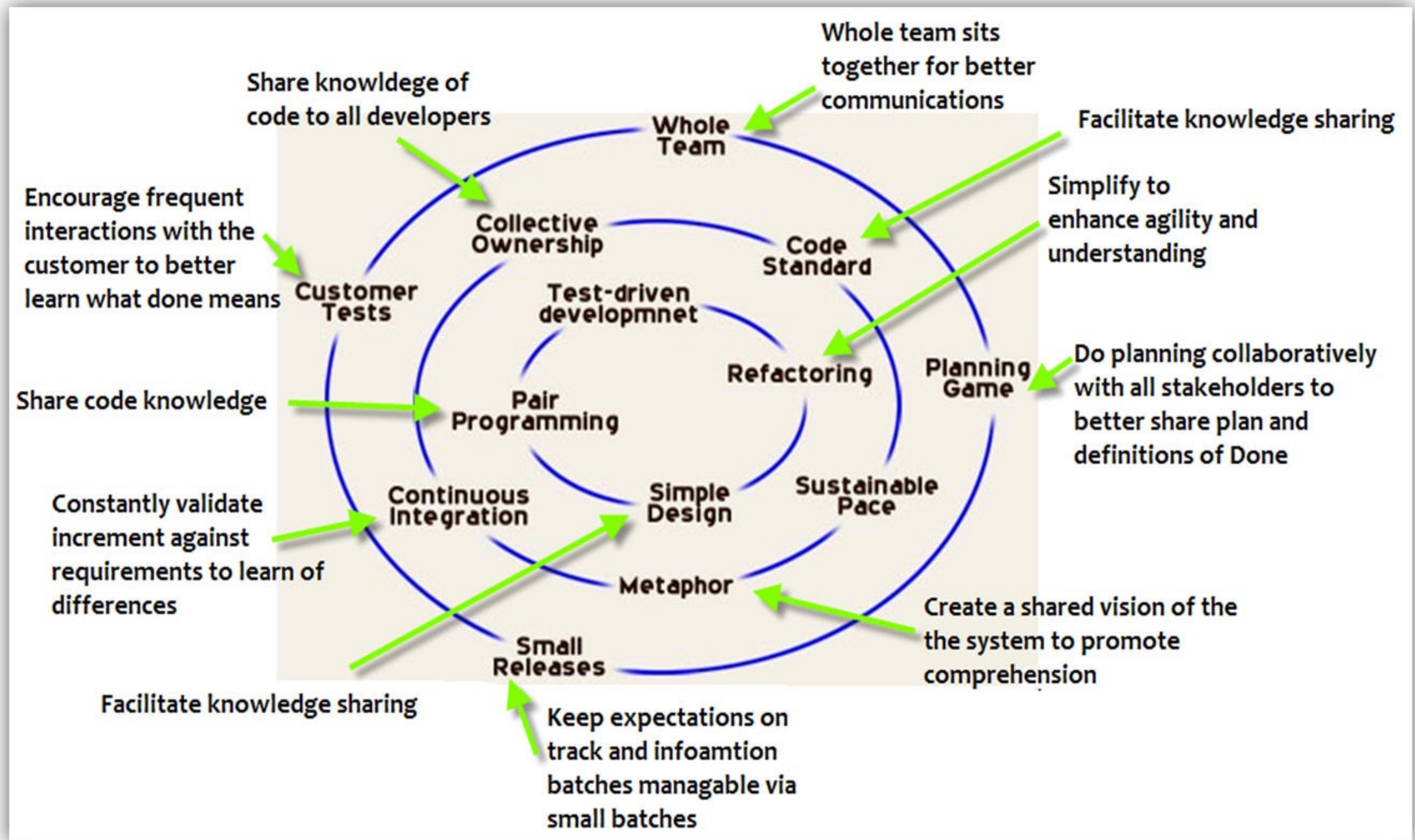
Planning phase

Iterate2release phase

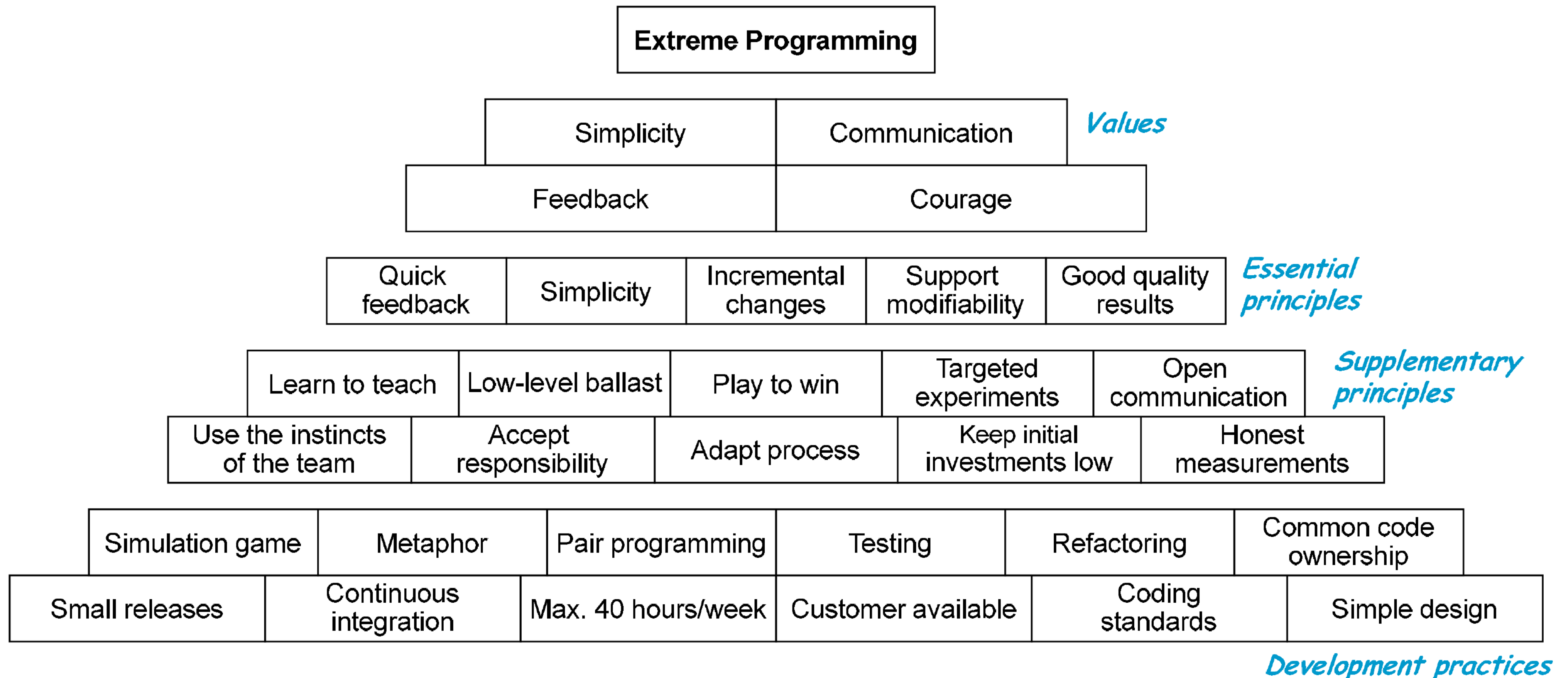
Production phase


XP Practices
Whole Team
Collective Ownership
Coding Standard
Test-Driven Development
Customer Tests
Pair Programming
Refactoring
Planning Game
Continuous Integration
Simple Design
Sustainable Pace
Metaphor
Small Releases
www.XProgramming.com

Knowledge Sharing in XP:



Valori, principi, pratiche agili in XP



Fonte: Rumpe, Agile modeling with UML, Springer 2017

Le pratiche di Extreme Programming

- I requisiti sono “user stories”
- Il “planning game”
- Piccoli rilasci
- Cliente On-site
- “Prima i test, poi il codice”
- La metafora di riferimento
- Integrazione continua
- Proprietà collettiva del codice
- Settimana di 40 ore di lavoro
- Uso sistematico di standard di codifica
- Programmazione di coppia
- Refactoring
- Progettazione semplice

I requisiti sono “user stories”

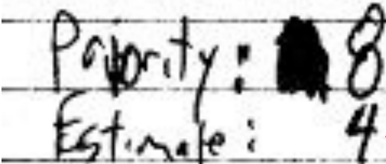
- User stories:
  - usate al posto di documenti dettagliati di specifica dei requisiti
  - scritte dai clienti: cosa si aspettano dal sistema
  - una storia è descritta da una o due frasi in testo naturale, con la terminologia del cliente (*no techno-syntax*)
  - utili per stimare i tempi/costi di un rilascio (*release planning)*

Esempi di user stories

- Uno studente può acquistare online un abbonamento di parcheggio
- Gli abbonamenti di parcheggio si possono pagare con carta di credito
- Gli abbonamenti di parcheggio si possono pagare con Paypal
- Un professore può inserire i voti online
- Uno studente può visualizzare l'orario delle lezioni
- Uno studente può ottenere una registrazione della lezione
- Uno studente può iscriversi ad un corso se soddisfa i prerequisiti
- La registrazione di un corso si può visualizzare mediante un browser

Esempio di scheda per user story

173. Students can purchase parking passes.



Story points

Formato preferito per le US

Come studente

voglio poter comprare on line un abbonamento mensile al parcheggio

per non tirare fuori soldi tutti i giorni

→Tipo di utente

→Obiettivo

→Motivazione

Il "Planning game"

- Le"**User Stories**" permettono di fare una pianificazione a breve termine
  - Una storia è una breve descrizione di qualcosa che vuole il cliente
  - La descrizione può essere arricchita da altre storie durante lo sviluppo
  - Le **priorità** tra le storie sono definite dal **cliente**
  - **Le risorse** necessarie (**story points**) e i rischi sono valutati dagli **sviluppatori**
- "The **Planning Game**"
  - Le storie di più alto rischio e priorità sono affrontate per prime, in incrementi "*time boxed*"
  - Il Planning Game si rigioca per ciascuna iterazione

Storie e iterazioni

La presenza del cliente

- Il cliente (un suo rappresentante) è sempre disponibile per chiarificare le storie e per prendere rapidamente decisioni critiche
- Gli sviluppatori non devono fare ipotesi: risponde direttamente il cliente
- Gli sviluppatori non devono attendere le decisioni del cliente
- La comunicazione “faccia a faccia” minimizza la possibilità di ambiguità ed equivoci

Il metodo Moscow per mettere in priorità le US

Moscow: acronimo per Must Should Could Won’t
(DEVE DOVREBBE POTREBBE NONPOTRÀ)

- Must: funzioni che DEBBONO esserci nel prodotto
- Should: funzioni che DOVREBBERO esserci
- Could: funzioni che POTREBBERO esserci
- Wont: funzioni che NON INSERIREMO nella versione attuale

Esempio

- L'utente potrà loggarsi con con username e passwd
- L'utente potrà registrare un nuovo account
- L'utente potrà vedere tutti i documenti Word inclusi nel file system
- L'utente potrà cancellare tutti i documenti Word
- L'utente potrà accedere ogni documento entro il file system
- L'utente potrà loggarsi con doppia autenticazione

Esempio

- L’utente DEVE loggarsi con con username e passwd
- L’utente DOVREBBE registrare un nuovo account
- L’utente DOVREBBE vedere tutti i documenti Word inclusi nel file system
- L’utente POTREBBE cancellare tutti i documenti Word
- L’utente DEVE accedere ogni documento entro il file system
- L’utente NON POTRÀ loggarsi con doppia autenticazione

In scope
for this timeframe

Out of scope
for this timeframe

(Project / Increment / Timebox)

Must Have

Typically no more than 60% effort

Should Have

Could Have

Typically around 20% effort

Won't Have this time

https://www.agilebusiness.org/page/ProjectFramework_10_MoSCoWPrioritisation

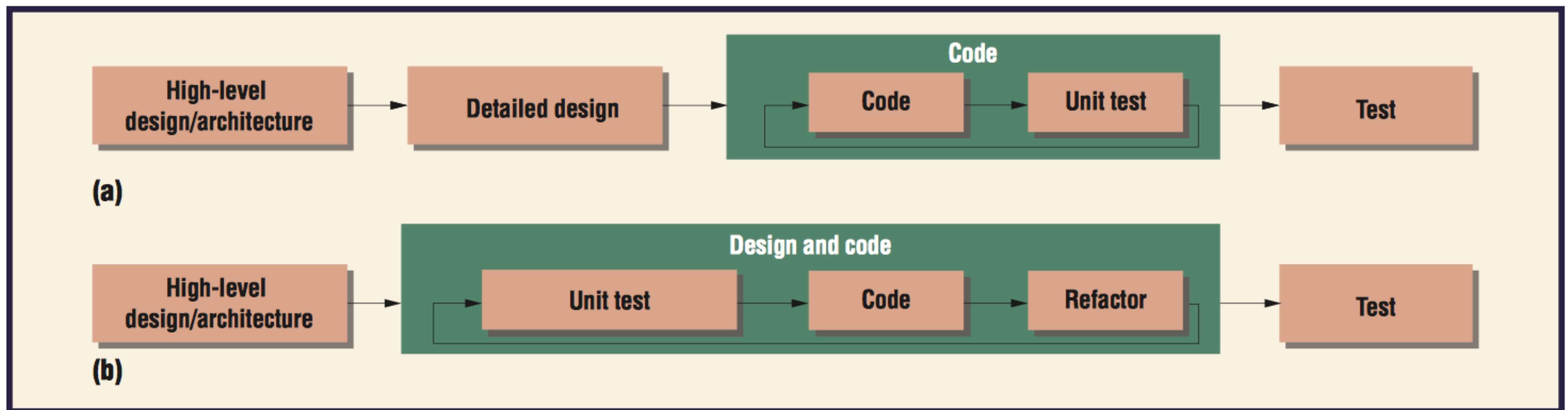
Sviluppo test-driven

Test-Driven Development (TDD)

- Scegliere una user story e definire i test prima del codice
- TDD è una tecnica di programmazione
- Automatizzare i test (es. usare xUnit)
- Deve girare tutto prima di proseguire con altro codice

- *Unit test* vs *acceptance test*

Test classico vs test driven



a) Testing classico b) test-driven design come tecnica di programmazione

Janzen & Saiedian, Does Test-Driven Development Really Improve Software Design Quality?, *IEEE Software*, march/April 2008

Customer tests

Test di accettazione

- Guidati dalle user stories
- Scritti col cliente
- Funzionano come "contratto"
- Misura del progresso

Esempio

Support technician sees customer's
history on screen at the start of a call

Esempio di user story su scheda

- Simulate a call with Fred's account number and verify that Fred's info can be read from the screen
- Verify that the system displays a valid error message for a non-existing account number
- Omit the account number in the incoming call completely and verify that the system displays the text "no account number provided" on the screen

Esempio di test scritto sul retro della user story

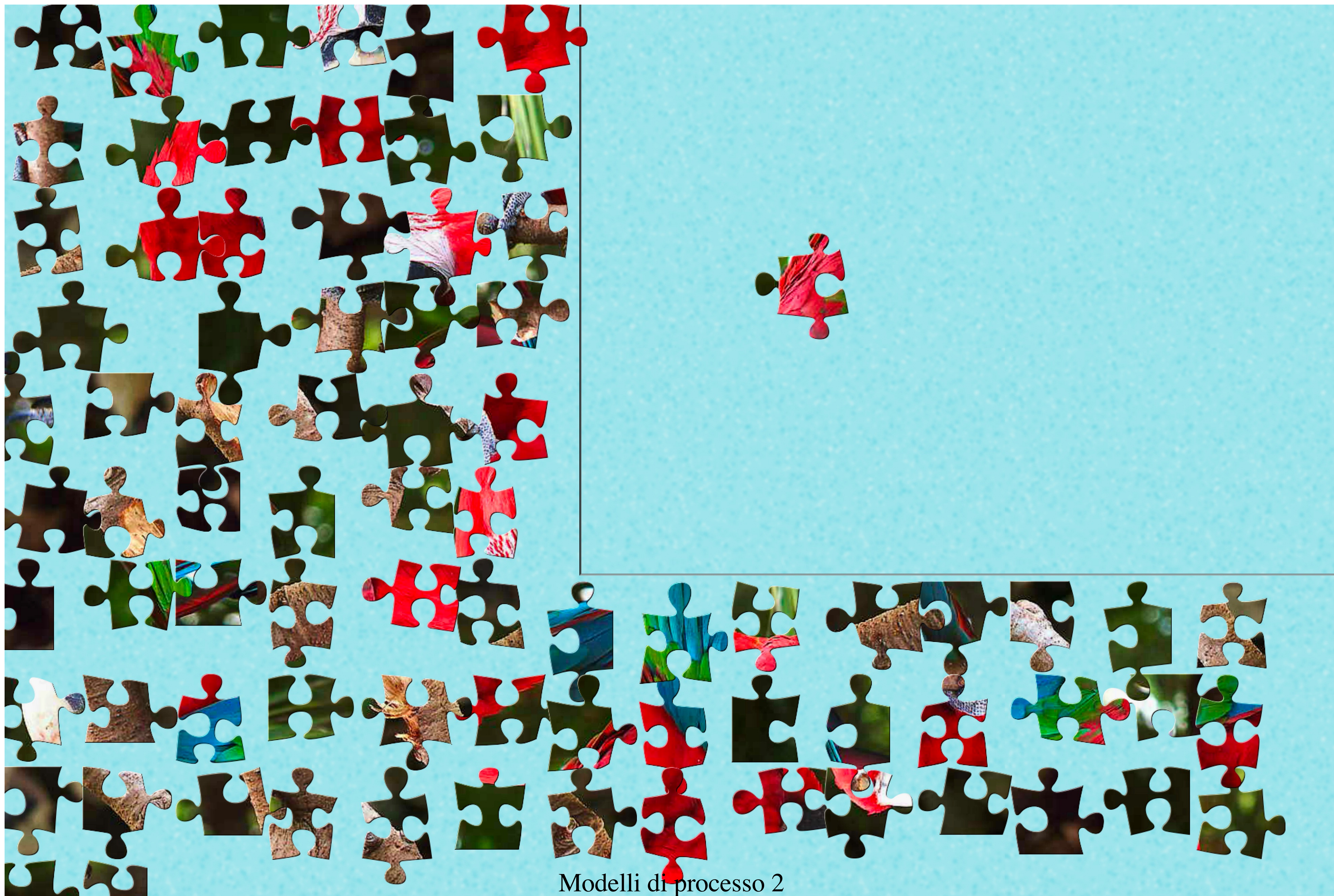
Piccoli rilasci

- Timeboxed (ovvero di durata “breve” prefissata)
- Minimali, ma comunque utili (microincrementi)
  - Mai cose come ‘implementare il database’
- Ottenere feedback dal cliente presto e spesso
- Eseguire il “planning game” dopo ciascuna iterazione
  - Si voleva qualcosa di diverso?
  - Le priorità sono cambiate?

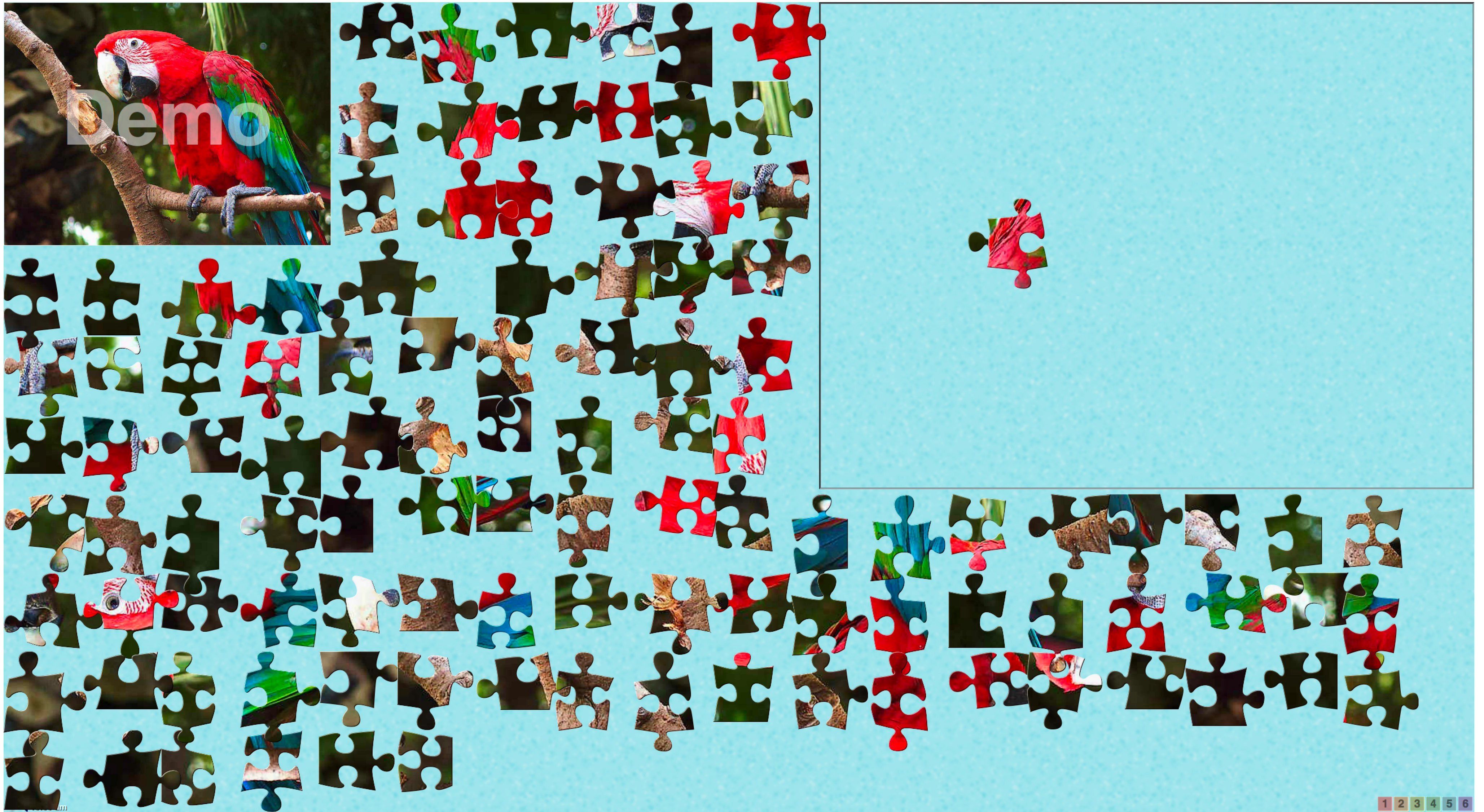
La metafora

- I progettisti XP sviluppano una visione comune di come funzionerà il programma, detta “**metafora del sistema**”
- Esempio: ”*questo programma funziona come uno sciame d’api, che cerca il polline a lo porta nell’alveare*” (sistema di information retrieval basato su agenti)
- Non sempre la metafora è poetica. In ogni caso il team deve usare un glossario comune di nomi di entità rilevanti per il progetto

http://agilemetaphors.com



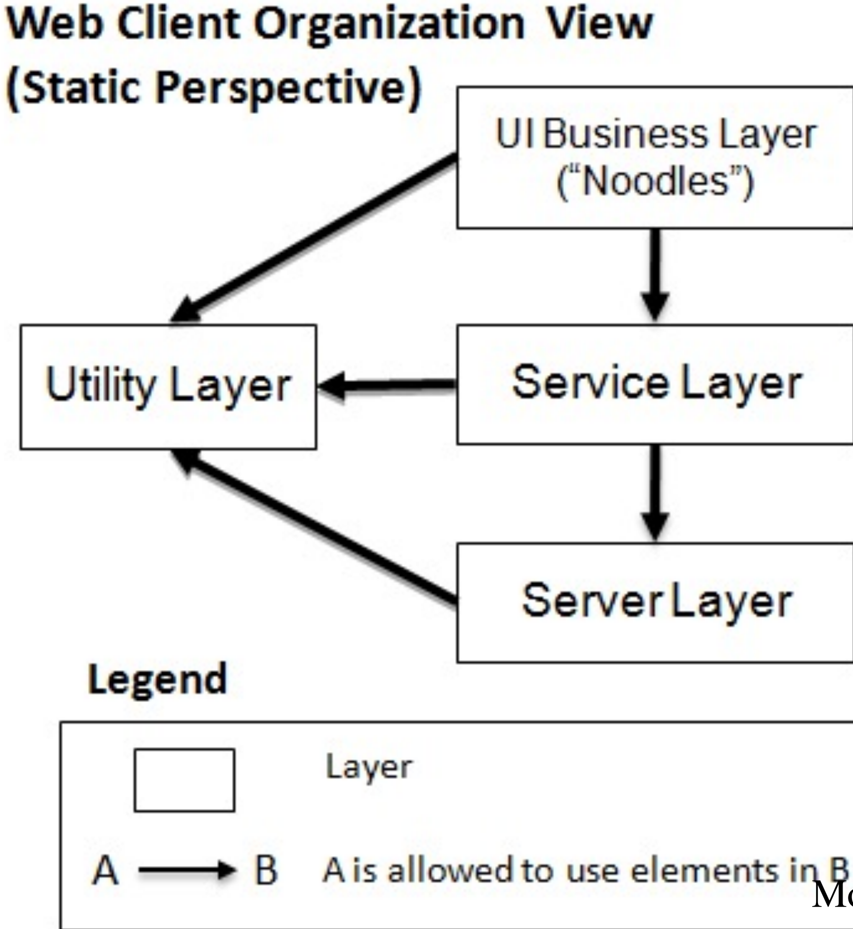
Modelli di processo 2



Modelli di processo 2

Esempio di metafora

Fonte: neverletdown.net/topics/architecture/



The "Bento Box"

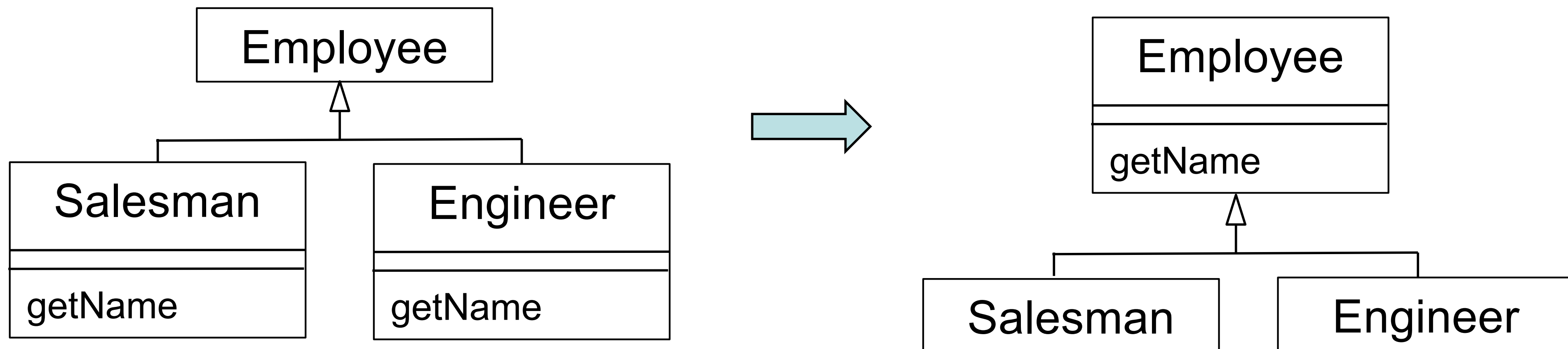
Progettare in modo semplice

- No Big Design Up Front (BDUF)
- "Fare la cosa più semplice che possa funzionare"
  - Includere la documentazione
- "You Aren't Gonna Need It" (YAGNI)
- Opzione: usare schede CRC
  (vedere le lezioni sui requisiti e sul design per un'introduzione alle schede CRC)

Rifattorizzare (refactoring)

- Refactoring: migliorare il codice esistente senza cambiarne la funzionalità
- Avere il coraggio di buttare via codice
  - Semplificare il codice
  - Rimuovere il codice ridondante
  - Cercare le opportunità di astrazione
- Il refactoring usa il testing per assicurare che con le modifiche non si introducano errori

Refactoring: esempio



Le sottoclassi hanno ciascuna metodi con risultati identici

Spostando il metodo comune nella superclasse, si elimina la ridondanza.
Eliminare le ridondanze nel codice è importante

Pair programming

La programmazione di coppia (pair programming) è tipica di eXtreme Programming (XP)

Con la programmazione di coppia:

- Due progettisti lavorano allo stesso compito su un solo computer
- Uno dei due, il driver, controlla tastiera e mouse e scrive il codice
- L'altro, il navigatore, osserva il codice cercando difetti e partecipa al brainstorming su richiesta
- I ruoli di driver e navigatore vengono scambiati tra i due progettisti periodicamente (es. a metà giornata)
- Le coppie cambiano ogni giorno: lo scopo è che tutti prendano confidenza col codice (vedi: proprietà collettiva del codice)

Pair programming migliora la qualità

	Progetto1: solisti	Progetto2: coppie
Dimensione (KLOC)	20	520
Team	4	12
Sforzo (mesi persona)	4	72
Produttività (KLOC/mp)	5	7.2
Produttività (KLOC/mpair)	n.d.	14.4
Difetti test unità	107 (5.34 difetti/KLOC)	183 (0.4 difetti/KLOC)
Difetti test integrazione	46 (2.3 difetti/KLOC)	82 (0.2 difetti/KLOC)

Fonte: Williams, Pair Programming Illuminated, AW, 2002

Integrazione continua

- La coppia scrive i test ed il codice di un task (= parte di una storia utente)
- La coppia esegue tutto il test di unità
- La coppia esegue l'integrazione della nuova unità con le altre
- La coppia esegue tutti i test di regressione
- La coppia passa al task successivo a mente sgombra (capita una o due volte al giorno)
- L'obiettivo è prevenire l' "*Integration Hell*"

Proprietà collettiva del codice

- Il codice appartiene al progetto, non ad un individuo
- Durante lo sviluppo tutti possono osservare e **modificare** qualsiasi classe
- Uno degli effetti del *collettive ownership* è che il codice troppo complesso non sopravvive a lungo
- Affinché tale strategia funzioni occorre l'"integrazione continua"

Passo sostenibile

- Kent Beck dice: “ . . . freschi e vogliosi ogni mattina, stanchi e soddisfatti ogni sera”
- Lavorare fino a tarda notte danneggia le prestazioni: uno sviluppatore stanco fa più errori, e alla lunga il gioco non vale la candela
- Se il progetto danneggia la vita privata dei partecipanti, alla lunga lo scotto lo paga il progetto stesso

Convenzioni di codifica

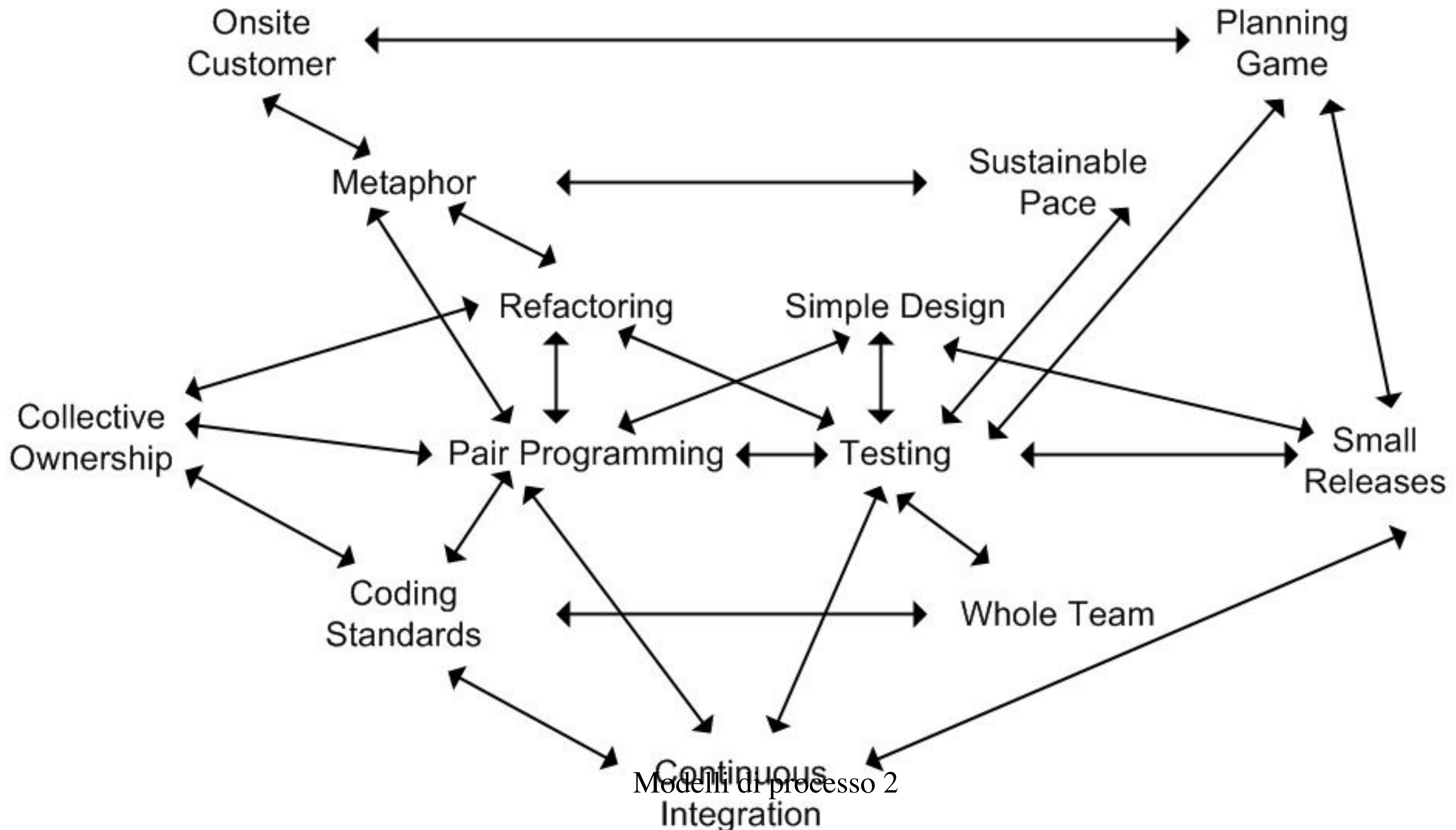
- Usare convenzioni di codifica
  - A causa del Pair Programming, delle rifattorizzazioni e della proprietà collettiva del codice, occorre poter addentrarsi velocemente nel codice altrui
- Commentare il codice
  - Priorità al codice che "svela" il suo scopo
    - Se il codice richiede un commento, riscrivilo
    - Se non puoi spiegare il codice con un commento, riscrivilo

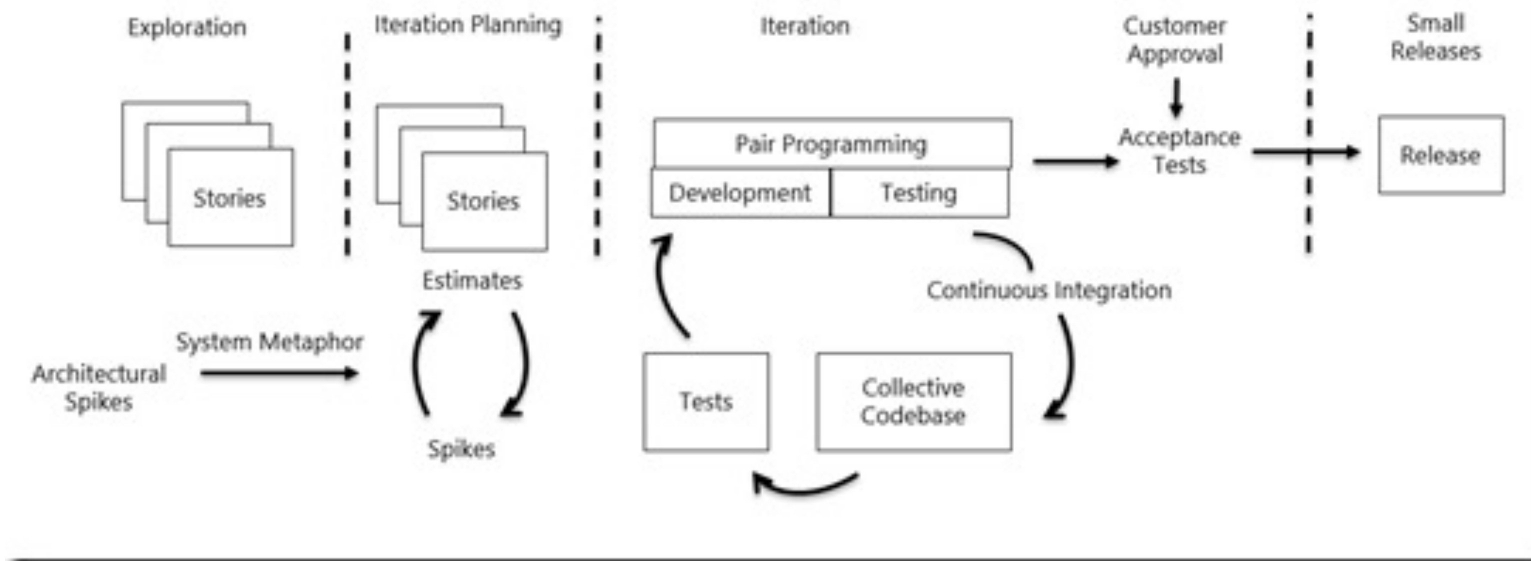
https://en.wikibooks.org/wiki/Computer_Programming/Coding_Style

http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vsent7/html7vxconcodingstandardscodereviews.asp

Relazioni tra le pratiche XP

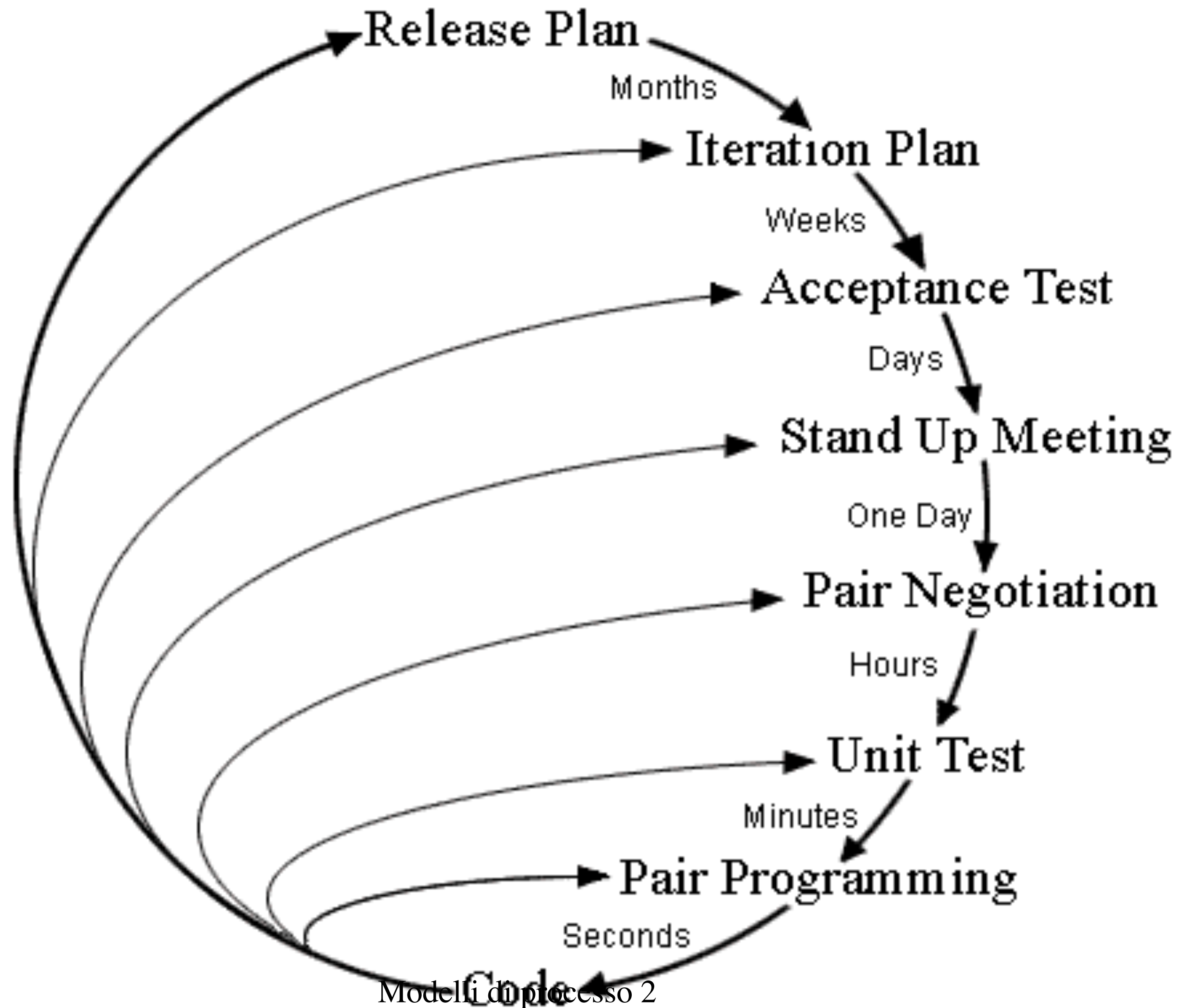


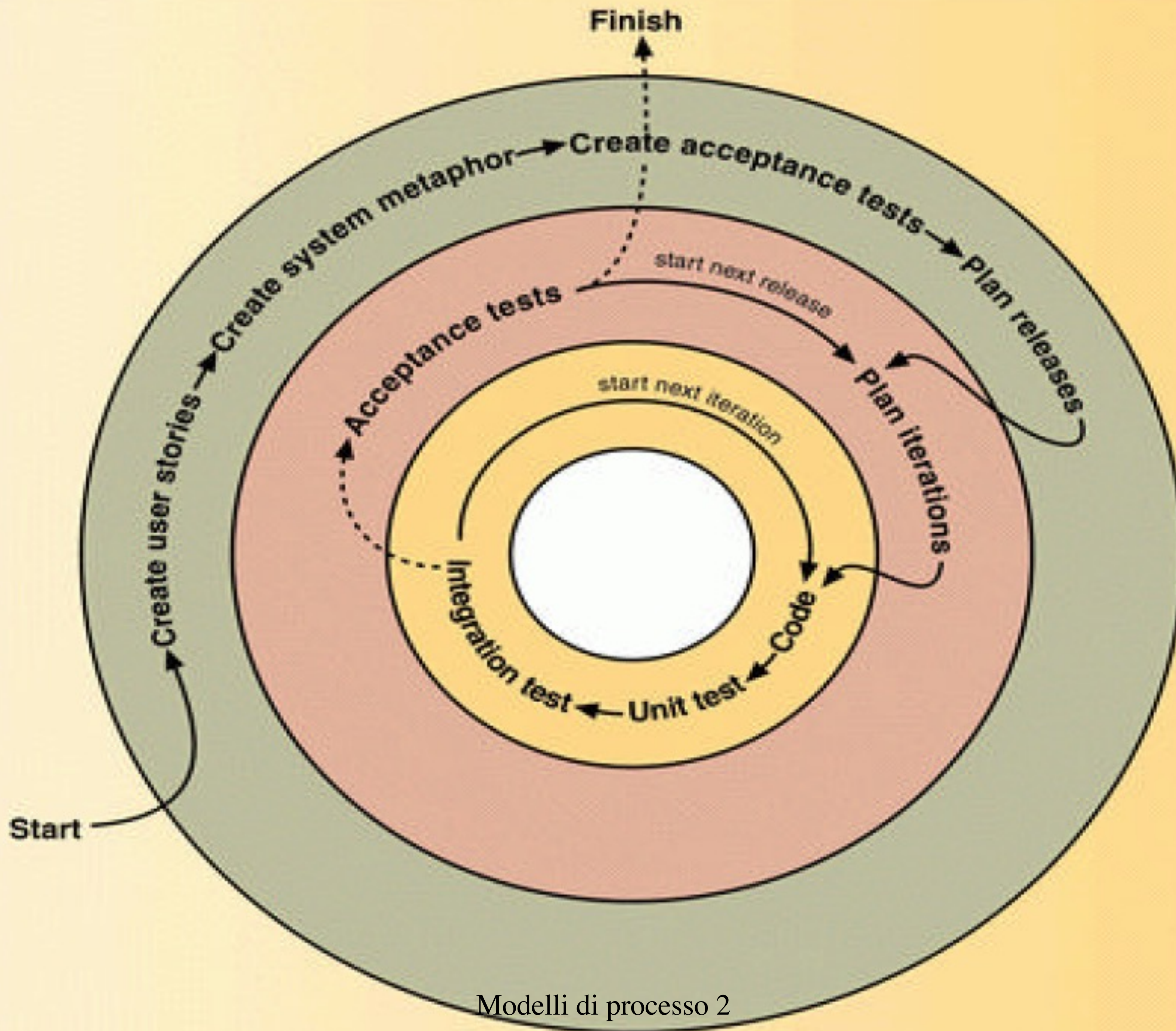

Extreme Programming (XP) at a Glance
Exploration
Iteration Planning
Iteration
Customer Approval
Small Releases
Stories
Stories
Pair Programming
Development
Testing
Acceptance Tests
Release
Estimates
Continuous Integration
System Metaphor
Architectural Spikes
Tests
Collective Codebase
Spikes

Il 13º Principio: La riunione in piedi

- Ogni giorno inizia con una riunione di 15 minuti
  - Tutti in piedi (così la riunione dura meno) in cerchio
  - Ciascuno a turno dice:
    - Cosa ha fatto il giorno prima
    - Cosa pensa di fare oggi
    - Quali ostacoli sta incontrando
  - Può essere il momento in cui si formano le coppie

Planning/Feedback Loops




Finish
Create system metaphor
Create acceptance tests
Plan releases
Create user stories
start next release
Acceptance tests
Plan iterations
start next iteration
Integration test
Code
Unit test
Start

Rischi del processo XP

- Il codice non viene testato completamente
- Il team produce pochi test utili per il cliente
- Il cliente non aiuta a testare il sistema
- I test “non funzionano” prima dell’integrazione
- I test sono troppo lenti
- Le storie sono troppo complicate
- Il sistema è troppo difettoso
- QA vuole il documento “Specifica dei requisiti”
- Il manager vuole la documentazione di sistema
- Il team è sovraccarico di compiti
- Il team deve affrontare scelte tecniche rischiose
- Alcuni membri (cowboy coders) ignorano il processo del team

Valutazioni dell'approccio XP

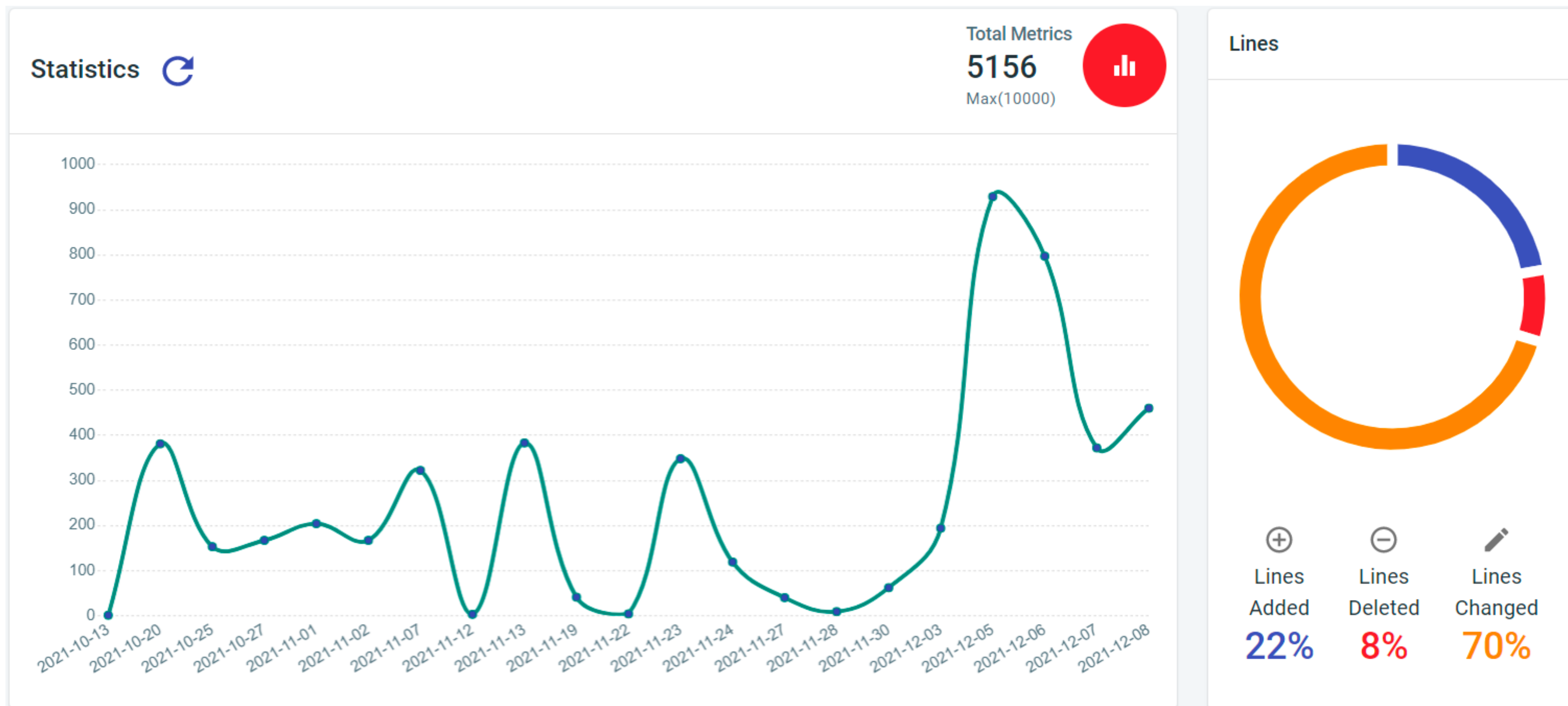
- Dall' industria abbiamo supporto anedottico "forte"
  - "Produciamo sw quasi senza difetti in metà tempo"
- Studi empirici
  - Le coppie producono codice di miglior qualità
    - 15% in più di casi di test superati (ovvero 15% meno errori)
  - Le coppie completano i loro compiti con minor sforzo individuale
    - Sforzo solo 15% in più (non 100%)
  - Molti programmatori all'inizio sono riluttanti a lavorare in coppia
    - Ma poi le coppie apprezzano di più il loro lavoro (92%)
    - Le coppie hanno più fiducia nei loro prodotti (96%)

Cockburn&Williams, The Costs and Benefits of Pair Programming

Extreme development

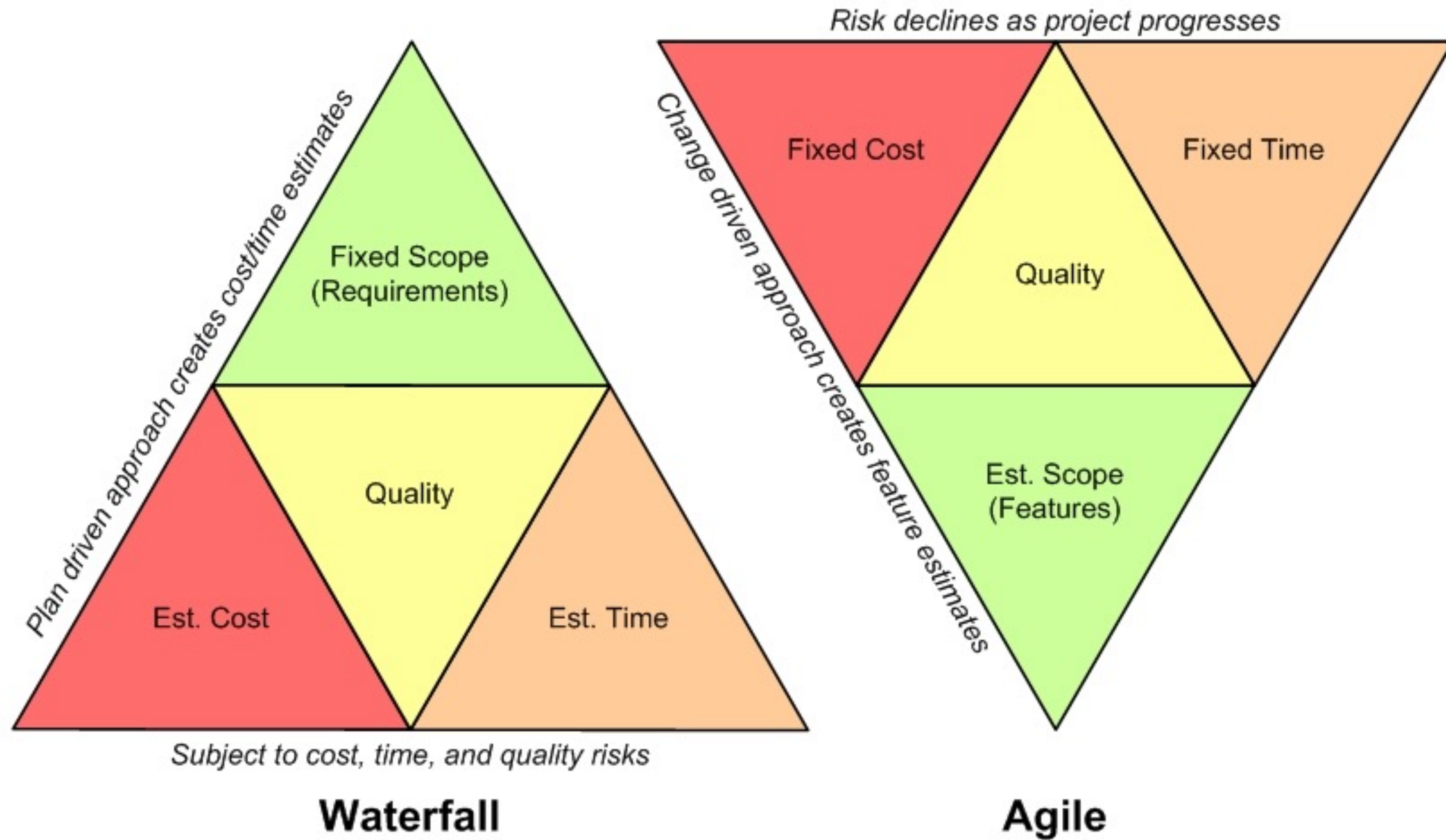
- Uso di soli strumenti open source
- Self tracking e team tracking
- Comunicazioni tracciate
- Uso di repository condiviso pubblico ma con controllo degli accessi

Self/team tracking

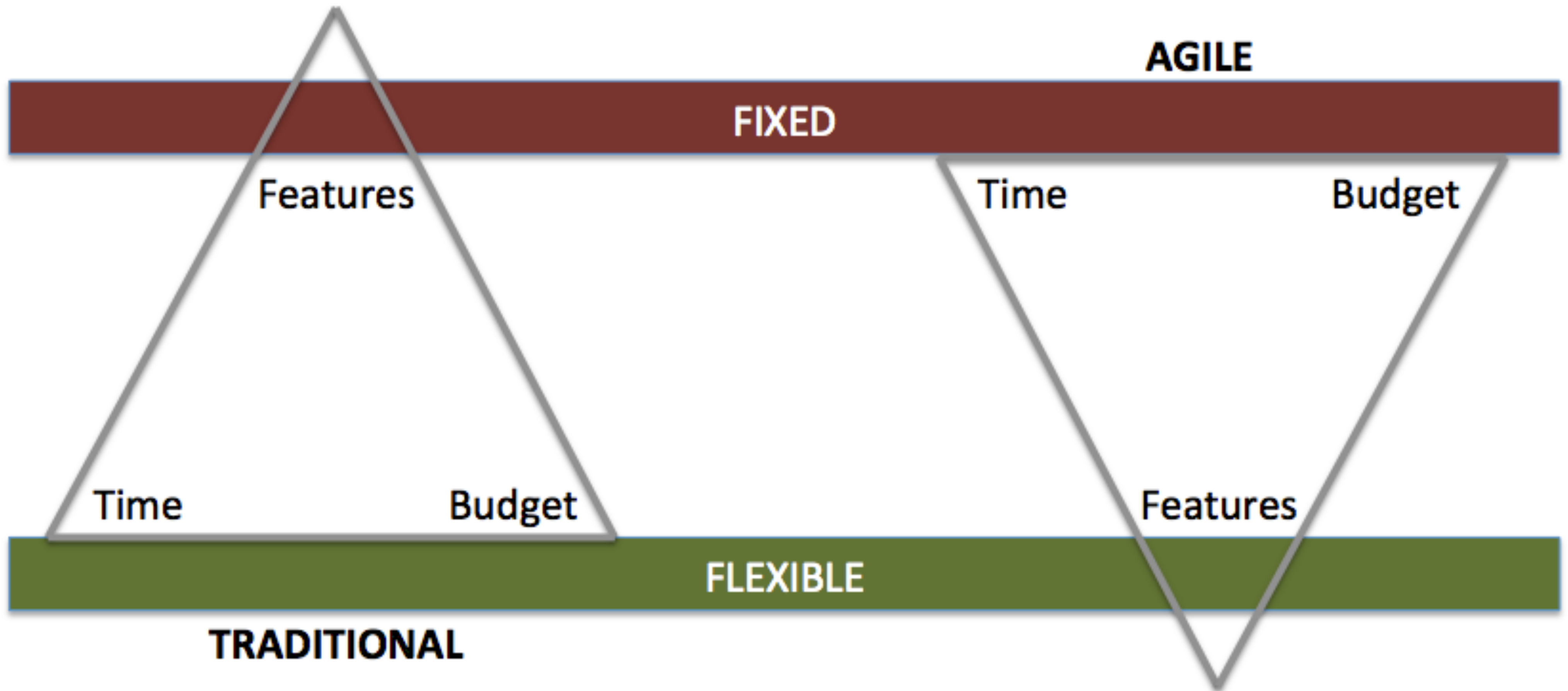


Agile o pianificato?

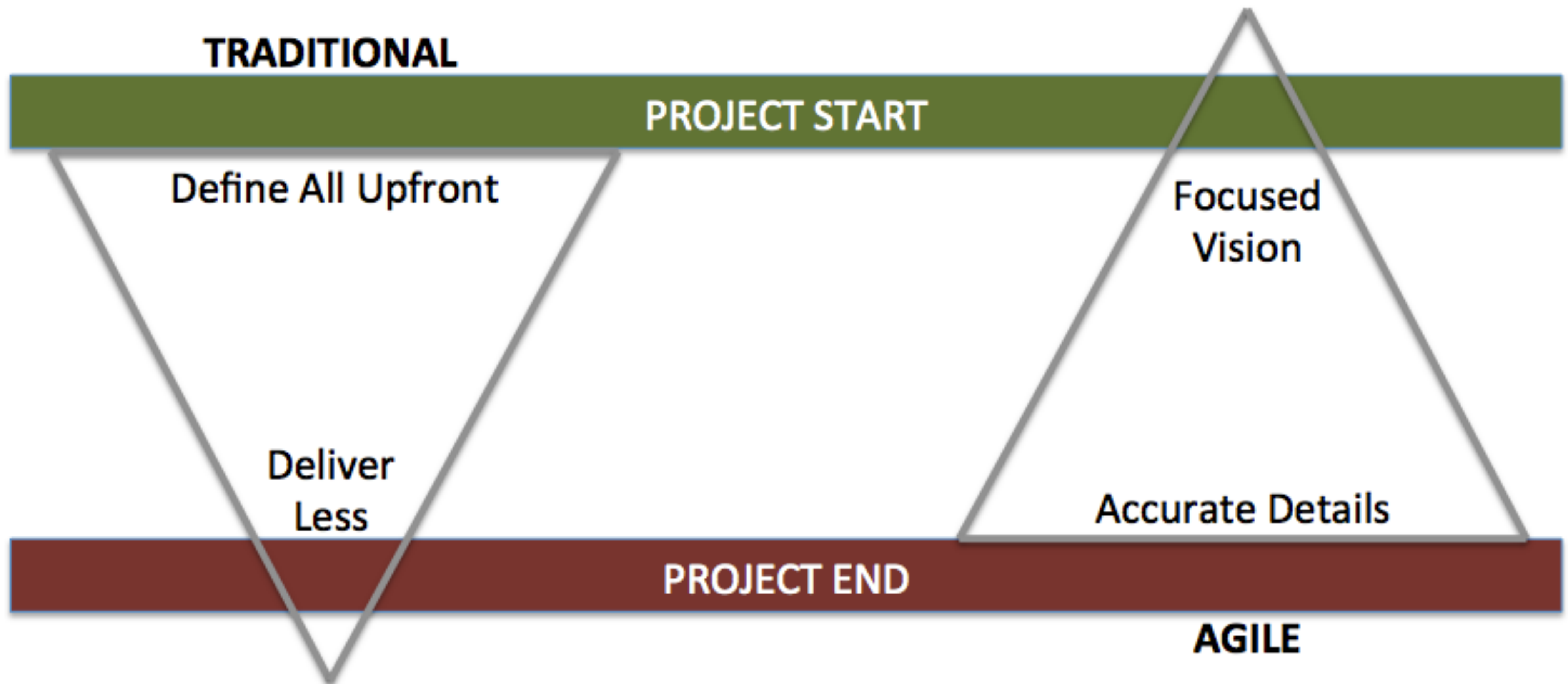
Fattore	Pro-agile	Pro-pianificato
Dimensione	Adatto a team che lavorano su prodotti sw “piccoli”. L’uso di conoscenza tacita limita la scalabilità	Adatto a grandi sistemi e team. Costoso da scalare verso i prodotti sw “piccoli”
Criticità	Utile per applicazioni con requisiti instabili (es. Web)	Utile per gestire sistemi critici e con requisiti stabili
Dinamismo	Refactoring	Pianificazione dettagliata
Personale	Servono esperti dei metodi agili; con Scrum devono essere “certificati”	Servono esperti durante la pianificazione
Cultura	Piace a chi preferisce la libertà di fare	Piace a chi preferisce ruoli e procedure ben definiti


Iron Triangle Paradigm Shift
Risk declines as project progresses
Plan driven approach creates cost/time estimates
Fixed Scope
(Requirements)
Quality
Est. Cost
Est. Time
Subject to cost, time, and quality risks
Waterfall
Change driven approach creates feature estimates
Fixed Cost
Fixed Time
Quality
Est. Scope
(Features)
Agile

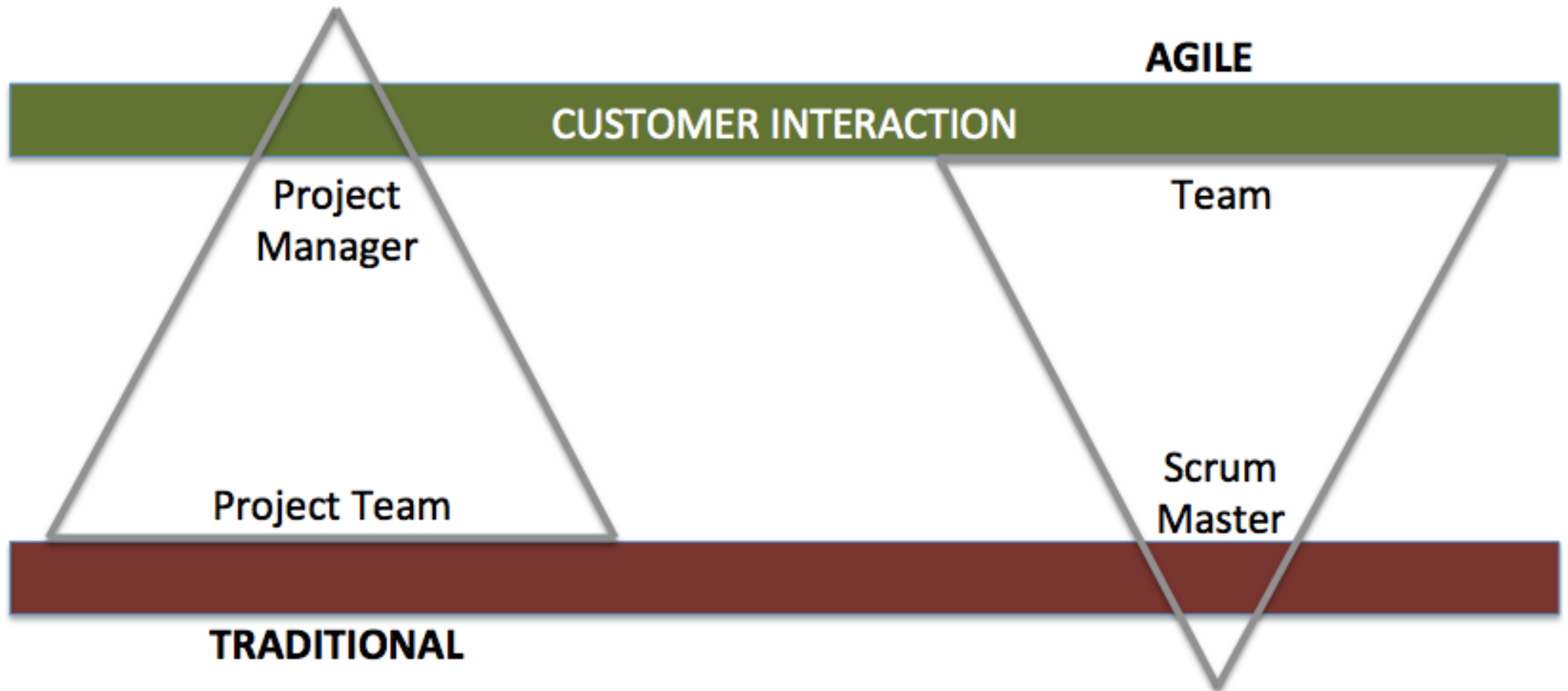
Il triangolo del Project Management -1



Il triangolo del Project Management -2



Il triangolo del Project Management -3



Cosa usano le grandi società?

https://newsletter.pragmaticengineer.com/p/project-management-in-tech 2021

Company	Is There a "Central" Methodology?	What Project Management "Methodology" Is Typically* Used for Engineering Projects?	Who Typically Leads Engineering Projects?
Amazon	No, teams can choose	Plan (6-pager)->Build (iterate)->Ship	Tech lead
Apple	No, teams can choose	Plan->Build (iterate)->Ship	Tech lead
Datadog	No, teams can choose	Plan (RFC)->Build (iterate)->Ship	Tech lead or an engineer
Facebook	No, teams can choose	Plan->Build (iterate)->Ship	Tech lead or an engineer
Google	No, teams can choose	Plan (Design Doc)->Build (iterate)->Ship	Tech lead or an engineer
Netflix	No, teams can choose	Plan->Build (iterate)->Ship	Tech lead or an engineer
Shopify	No, teams can choose	GSD (Get Shit Done, 6-week cycles)	Tech lead or an engineer
Spotify	No, teams can choose	Plan->Build (iterate)->Ship	Tech lead or an engineer
Uber	No, teams can choose	Plan (ERD)->Build (iterate)->Ship	Tech lead or an engineer

newsletter.pragmaticengineer.com

Cosa usano le grandi società?

https://newsletter.pragmaticengineer.com/p/project-management-in-tech 2021

Is There a "Central" Methodology?	*Most Common Project Management Methodologies (In Order)*	*Who Typically Leads Engineering Projects?*
Big Tech & Public Tech Companies		
- Teams can choose how they work (common) - Suggested methodology, but teams can choose (less frequent)	*1. Plan->Build (iterate)->Ship 2. No "formal" methodology*	*- Tech lead - An engineer on the team*
***Venture-funded scaleups** (Series B & above)*		
- Suggested methodology, but teams can choose (common) - Teams expected to follow specific a methodology (less frequent)	*1. Plan->Build (iterate)->Ship 2. No "formal" methodology 3. Kanban 4. Scrum*	*- Tech lead - An engineer on the team*
***Venture-funded startups** (up to Series A)*		
- Teams expected to follow specific a methodology (common) - Suggested methodology, but teams can choose (less frequent)	*1. Plan->Build (iterate)->Ship 2. Scrum 3. Kanban*	*- An engineer on the team - Product manager*
Non-venture funded tech companies		
- Teams expected to follow specific a methodology (mostly) - Suggested methodology, but teams can choose (less frequent)	*1. Scrum 2. Others (Kanban, SAFe, Scaled Agile)*	*- Tech lead - Dedicated project manager - An engineer on the team*
Large, non-tech companies		
- Teams expected to follow specific a methodology (mostly) - Suggested methodology, but teams can choose (rarely)	*1. Scrum 2. SAFe 3. Others (Plan->build->ship, Kanban)*	*It varies: - A dedicated project manager - Scrum master - Product manager/owner - Tech lead*
Consultancies		
- Teams expected to follow specific a methodology (mostly) - Suggested methodology, but teams can choose (rarely)	*1. Scrum 2. No "formal" methodology*	*A dedicated project manager*

Conclusioni sui modelli agili

- Il cliente o un suo rappresentante (Product Owner) lavora col team
- Il team accetta le modifiche ai requisiti, cioè delle user stories, in ogni fase dello sviluppo
- Lo sviluppo è costantemente guidato dai test e dall'accettazione del PO
- Delivery rapide e frequenti
- Design semplice, refactoring sistematico, proprietà collettiva del codice e integrazione continua
- Coinvolgimento del cliente e responsabilizzazione dei membri del team
- Analisi retrospettiva, per migliorare ad ogni iterazione
- Modelli agili adatti per prodotti di piccole/medie dimensioni

Anti-metodologia?

Il movimento Agile non è contrario alle metodologie, anzi molti di noi vogliono ridare credibilità alla parola *metodologia.*

Vogliamo ripristinare un equilibrio.

Abbracciamo la modellazione, ma non per archiviare qualche diagramma in un polveroso repository aziendale

Abbracciamo la documentazione, ma non centinaia di pagine di tomi mai aggiornati e raramente usati.

Pianifichiamo, ma riconosciamo i limiti della pianificazione in un ambiente turbolento.

Jim Highsmith, *History: The Agile Manifesto*

Oltre l'Agile

Come aspiranti artigiani del software stiamo spostando l'asticella dello sviluppo professionale del software praticandolo e aiutando altri ad apprendere l'arte. Noi apprezziamo:

- Non solo il software funzionante, ma anche **il software ben fatto**
- Non solo rispondere al cambiamento, ma anche **aggiungere valore costantemente**
- Non solo gli individui e le interazioni, ma anche **una comunità di professionisti**
- Non solo la collaborazione col cliente, ma anche **una partnership produttiva**

Cioè nel cercare gli elementi di sinistra abbiamo capito che quelli di destra sono indispensabili.

http://manifesto.softwarecraftsmanship.org

Letture raccomandate

- Cohn e Ford, Introducing an agile process to an organization, *IEEE Computer*, 2003
- Bohem e Turner, Using Risk to Balance Agile and Plan-Driven Methods, *IEEE Computer*, 2003

Riferimenti

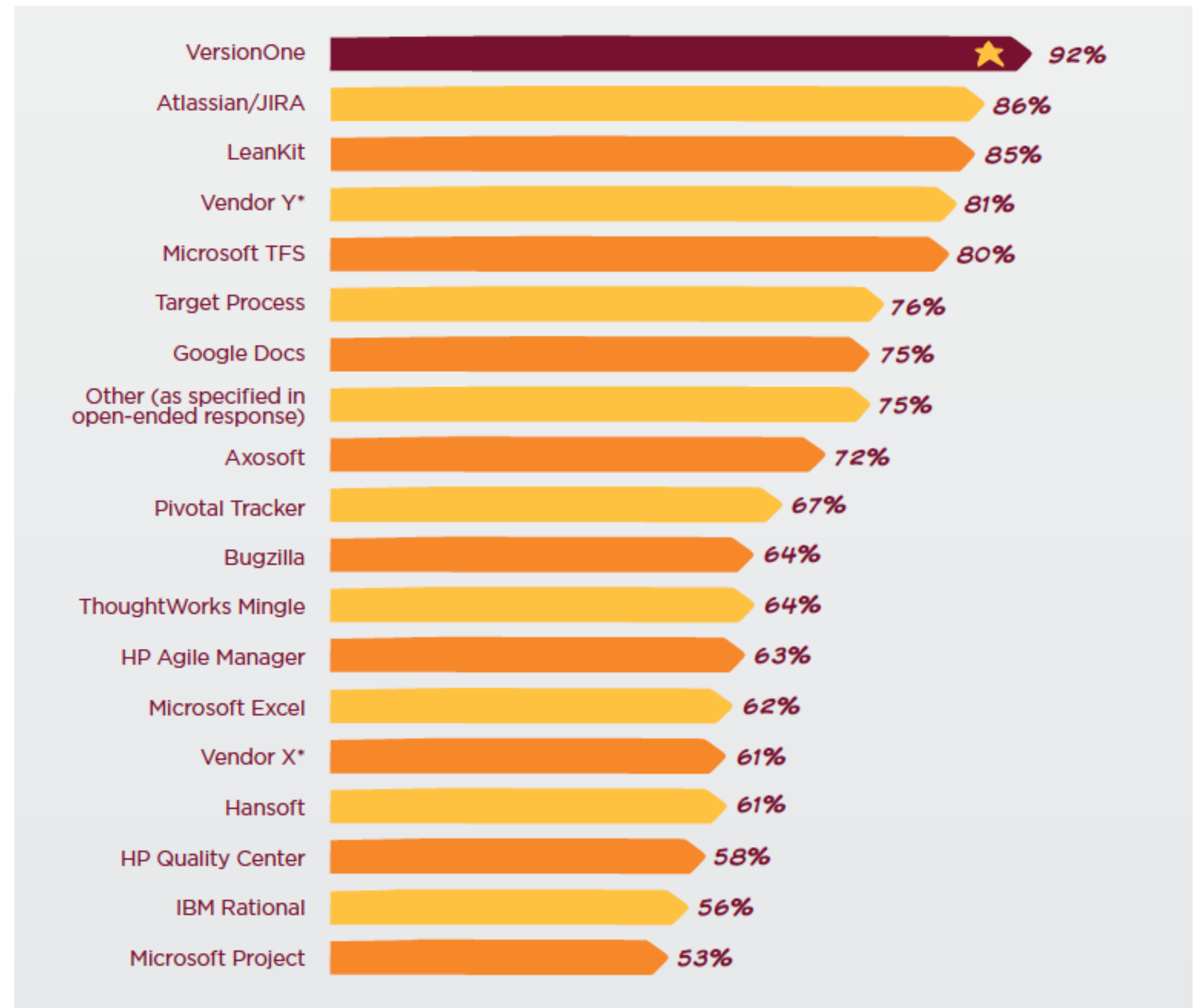
- Beck e altri, Manifesto for Agile development, 2001 `agilemanifesto.org`
- Beck & Andres, *Extreme Programming Explained: Embrace Change*, AW 2004
- Wilson, *Building software together*, https://buildtogether.tech

Siti e blog

- Extreme Programming `www.extremeprogramming.org`
- Uncle Bob’s blog `blog.cleancoder.com`
- `martinfowler.com/articles/on-pair-programming.html`
- `xp123.com/articles/`

Strumenti

- Trello/Taiga
- Slack/Mattermost
- Ant, XDoclet, JUnit, Cactus, Maven
- Atlassian Jira
- agiletrack.net
- www.planningpoker.com
- https://www.collab.net



Pubblicazioni di ricerca

- International Conference on Agile Software Development (XP)

Domande?