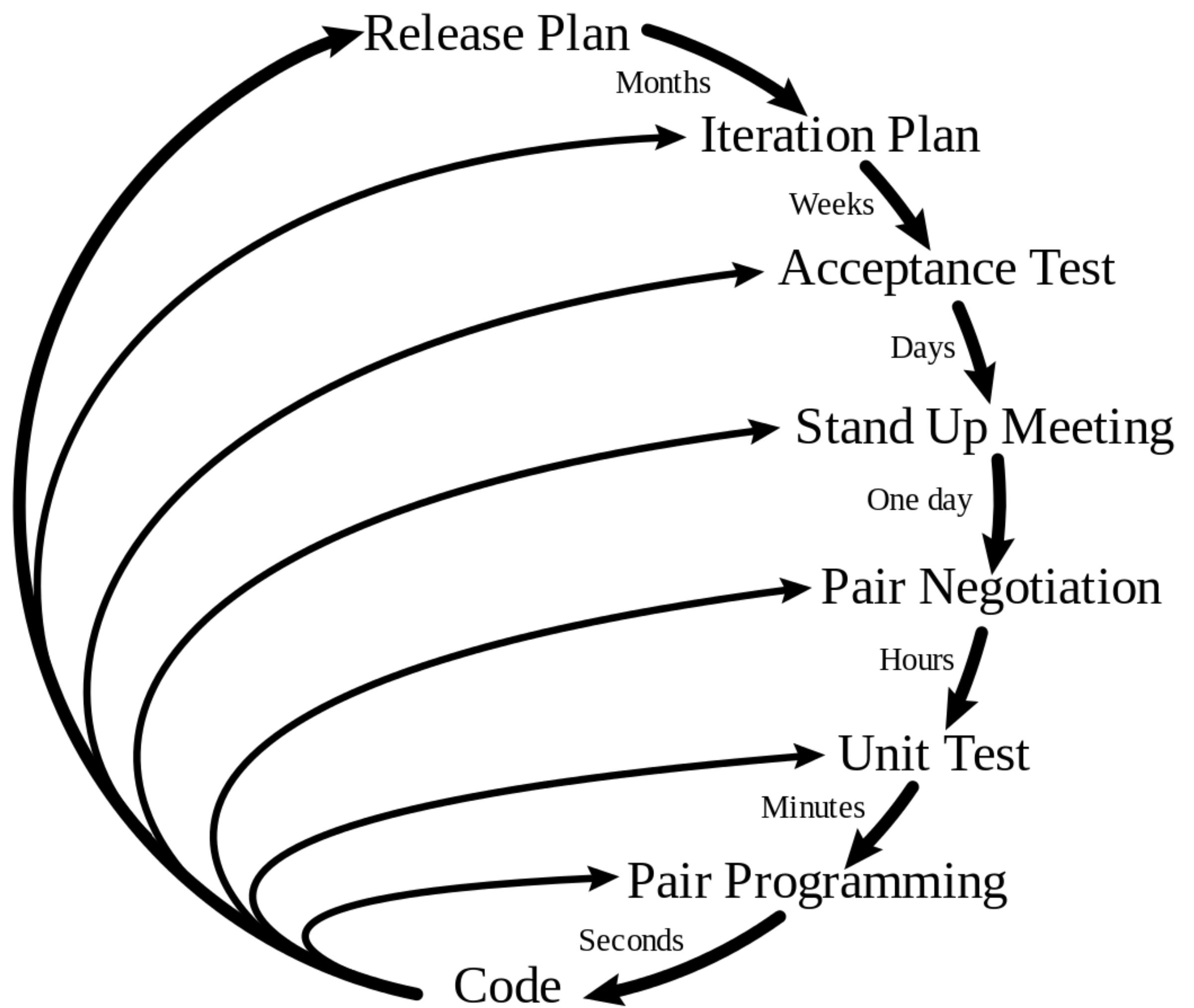


eXtreme Programming

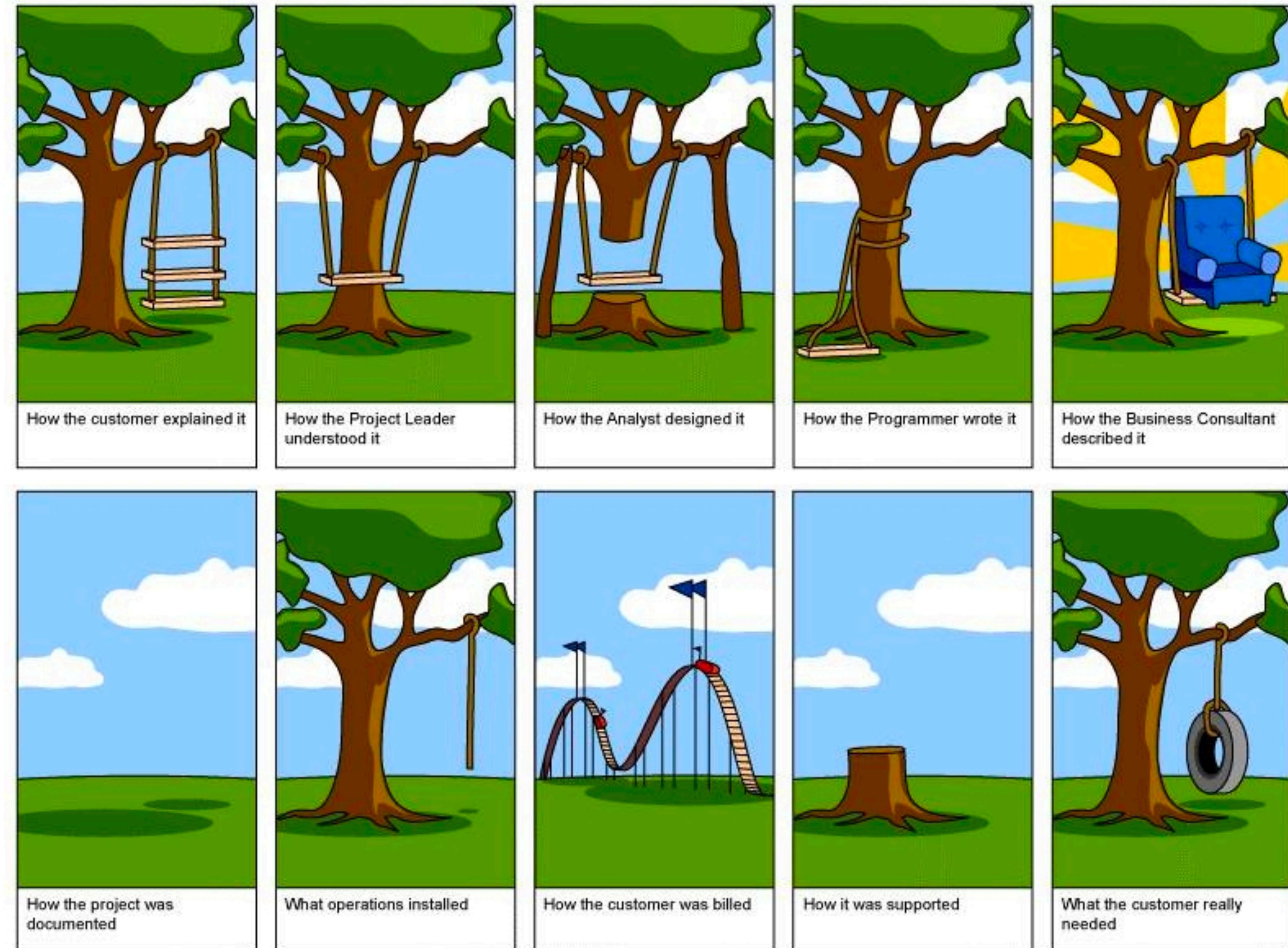
Planning/Feedback Loops



Perché eXtreme Programming?

CHAOS Report, Standish Group, 1994

- 31.1% dei progetti non vengono mai realizzati.
- 52.7% dei progetti costerà il 189% del costo preventivato.
- 16.2% dei progetti viene completato nei tempi e nei costi stabiliti (nelle aziende grandi la percentuale stimata è 9%).
- I progetti completati hanno approssimativamente il 42% delle caratteristiche e delle funzionalità richieste.



Modelli Agile vs Modelli Tradizionali

Individui e interazioni

Software funzionante

Collaborazione col cliente

Rispondere al cambiamento

su

su

su

su

Processi e strumenti

Documentazione esaustiva

Negoziazione di un contratto

Seguire un piano

eXtreme Programming

XP è un metodo di sviluppo nato ufficialmente in un progetto sviluppato da Kent Beck, dopo aver lavorato per parecchi anni con Ward Cunningham.

“Tutto cambia nel software. I requisiti cambiano. La progettazione cambia. Gli aspetti commerciali cambiano. La tecnologia cambia. I componenti del team cambiano. Il problema non è il cambiamento, di per sé, perché i cambiamenti avverranno; il problema, piuttosto, è l’inabilità di far fronte ai cambiamenti quando essi avvengono.”

Kent Beck

“XP è un metodo per sviluppare software: in maniera leggera, efficiente, low-risk, flessibile, prevedibile e scientifica.”

Kent Beck

Molte delle pratiche sono in uso da decadi. In sintesi le innovazioni sono:

- Mettere le pratiche sotto un “unico ombrello”.
- Essere sicuri che possano essere praticabili in ogni contesto.
- Essere sicuri che il grado ogni individuo coinvolto nel progetto abbia supporto per le pratiche.



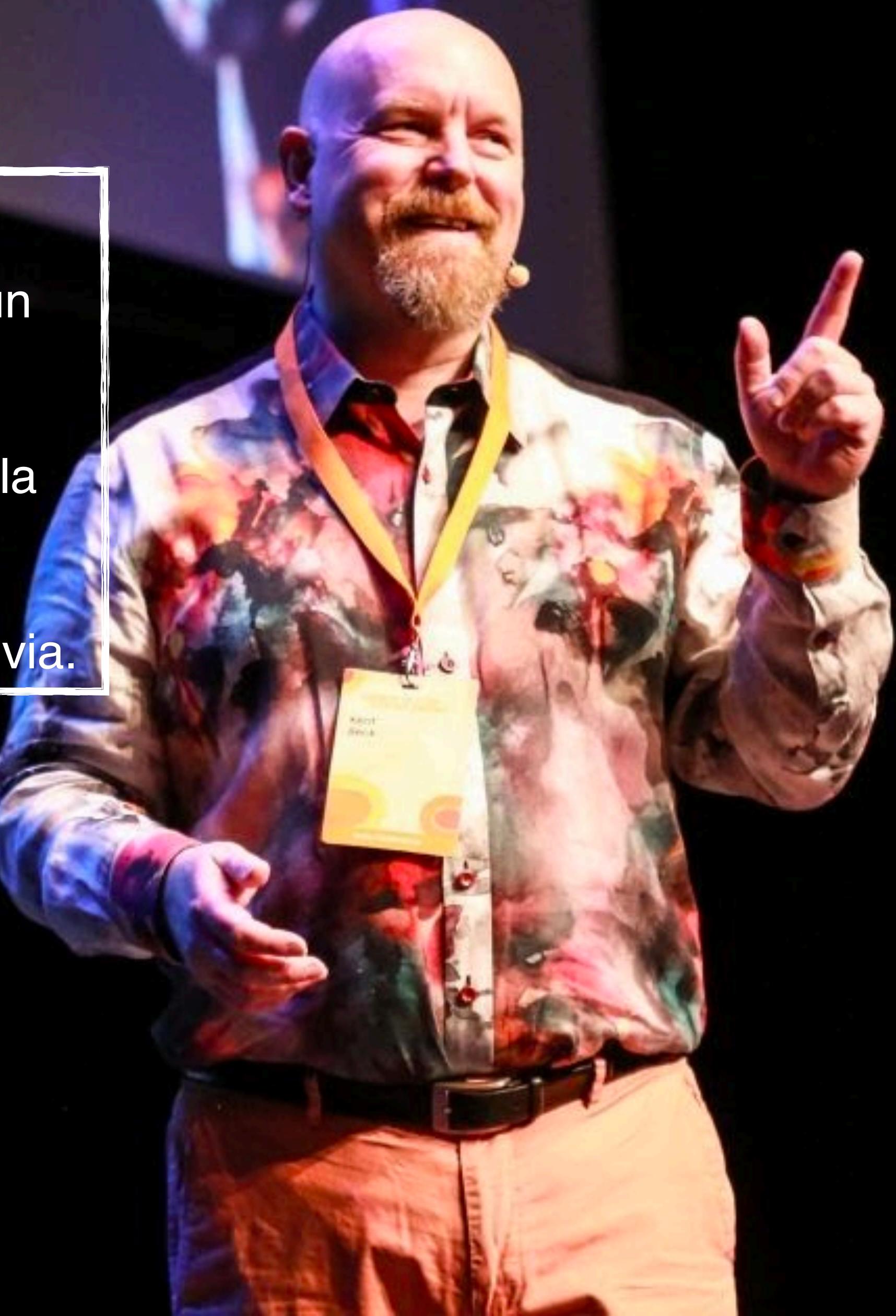
XP: Caratteristiche

- È rapido, concreto, e fornisce continui feedback.
- È un approccio di pianificazione incrementale che dura per tutto il progetto.
- L'implementazione di nuove funzionalità è flessibile rispetto ai cambiamenti
- Si basa su test sviluppati da programmatore e clienti per consentire al software di evolvere rapidamente e catturare difetti il prima possibile.
- Test, comunicazione verbale e codice sorgente sono gli strumenti per comunicare la struttura del sistema e il suo scopo.
- Si basa a breve termine sull'istinto dei programmatore e a lungo termine sull'interesse del progetto.



Test e qualità

- Aumentare la qualità significa completare il progetto in meno tempo.
- Un gruppo abituato a fare test intensivi e a standard di codifica svilupperà un progetto più velocemente.
- La qualità del progetto rimarrà assicurata grazie ai test. Questo aumenterà la confidenza nel codice e, quindi, faciliterà il cambiamento.
- Questo permetterà alle persone di programmare più velocemente ... e così via.

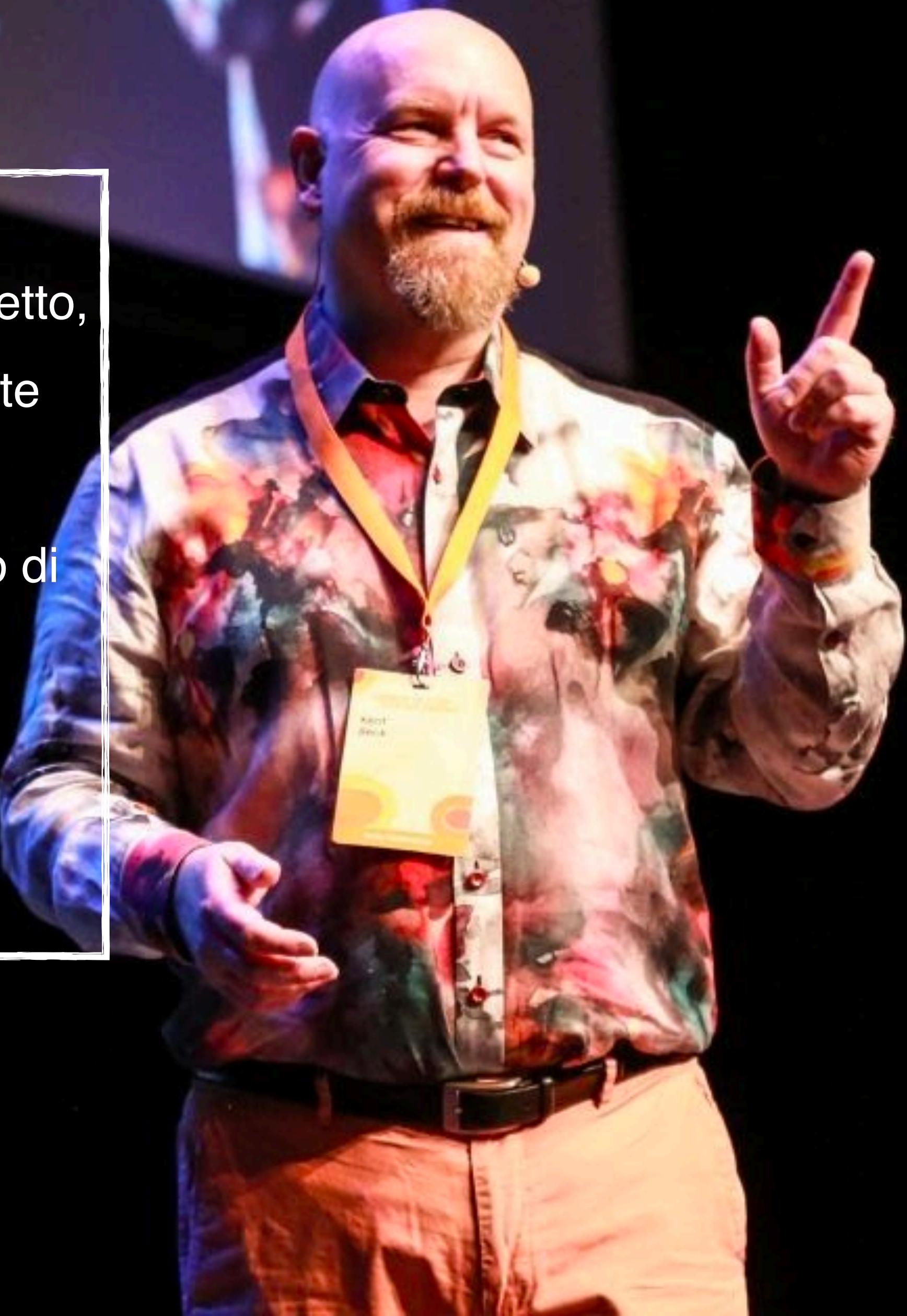


Test e qualità

L'altra faccia della medaglia è la tentazione di sacrificare la qualità interna del progetto (percepita dai programmati) per ridurre il tempo di rilascio del progetto, ritenendo che la qualità esterna (percepita dai clienti) non sarà eccessivamente deteriorata.

Si ignora il principio “ognuno lavora meglio se gli è permesso di fare un lavoro di qualità”. Ignorare questo significa:

- demoralizzato in gruppo nel lungo termine
- rallentare il progetto
- perdere più tempo di quello che si sperava di guadagnare



Come la F1...



Sviluppare software facendo piccole correzioni con lo sterzo prestando attenzione a quello che accade intorno.

- E' necessario controllare lo sviluppo del software facendo molti piccoli cambiamenti.
- Non bisogna effettuare pochi grandi aggiustamenti.
- Questo significa che sono necessari continui feedback.
- Per apportare correzioni ad un costo ragionevole.

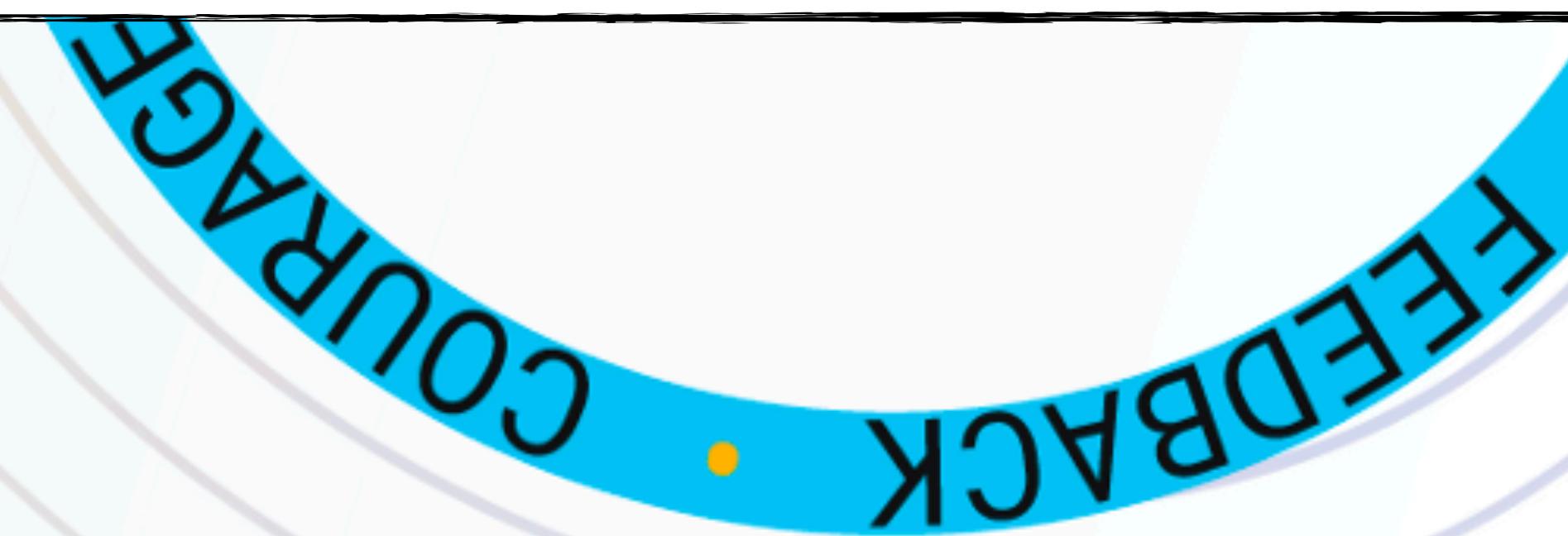
Quattro Valori



Uno degli scopi è favorire la giusta comunicazione tra le persone coinvolte in un progetto.

Bisogna favorire la comunicazione mediante pratiche di breve durata. Esempi:

- User Stories
- Pair programming
- Task estimation
- Iteration planning
- Design sessions



Quattro Valori

Sviluppare le funzionalità più facili rispetto al business value.

E' meglio fare una cosa semplice oggi ed eventualmente cambiarla domani.

Semplicità e comunicazione devono mutuamente supportarsi.

Quattro Valori

La scala del feedback varia

- minuti e giorni (unità);
- settimane e mesi (user stories).

Come migliorare il feedback?

- Piccole iterazioni
- Rilasci frequenti
- Pair programming
- Code review costante
- Continuous integration
- Automated unit tests

P
values

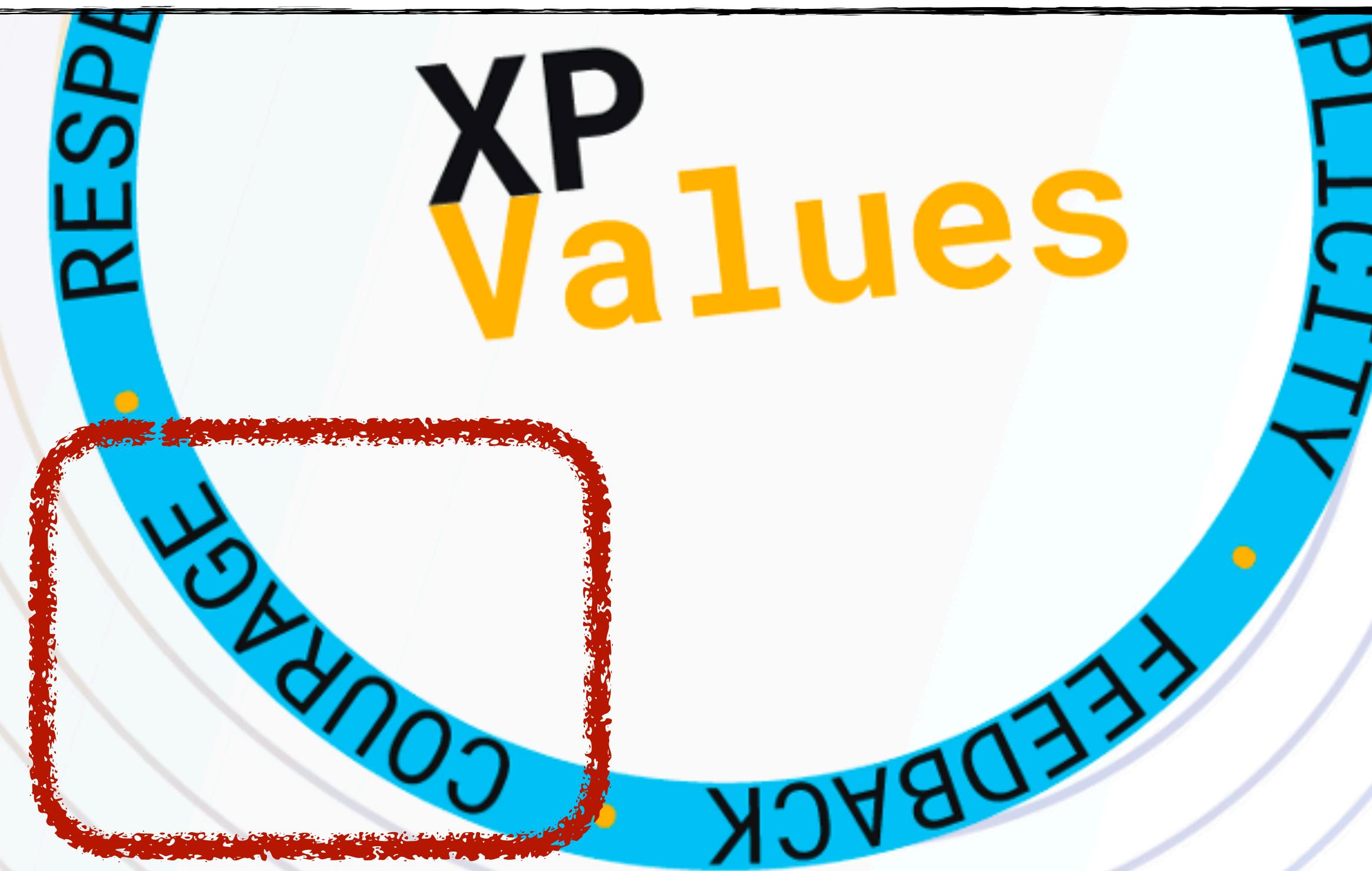
Quattro Valori

Coraggio di buttare tutto quello che hai fatto.

Coraggio di dire che il codice realizzato da te o da un altro programmatore potrebbe essere fatto meglio.

Seguire tutte le possibili alternative di progettazione, e successivamente scegliere la più semplice.

Il coraggio deve sempre condurre ad una soluzione più semplice.



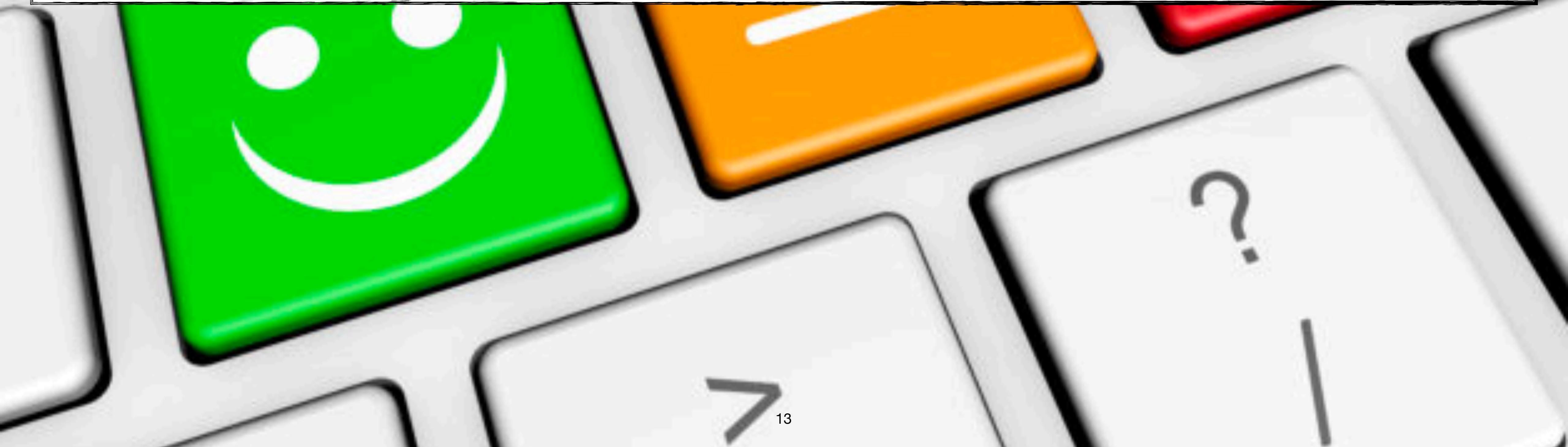
Cinque Principi di Base: Fornire Feedback

Il tempo fra un'azione ed il suo feedback è un aspetto critico da imparare (dalla psicologia).

Gli sviluppatori forniscono ed ottengono feedback prima possibile, li interpretano ed agiscono sul sistema in funzione.

Servono giorni e non mesi per ottenere feedback dai clienti.

Servono minuti e non settimane per ottenere feedback dagli sviluppatori.



Cinque Principi di Base: Assumi la Semplicità



Occam's Razor: No propositio

Progetta solo quello che è importante per quello che stai facendo.

“Si può risparmiare il 98% del tempo e dedicarlo a cose realmente più complesse.” (cit. Beck)

Pianifica per il futuro e progetta per il riuso.

Fai cose di qualità (tests, refactoring, communication) e confida nel fatto che la complessità venga dopo.

The explanation of a
problem depends on, the better the
explanation.

(William of Occam)

Cinque Principi di Base: Apporta Cambiamenti Incrementali

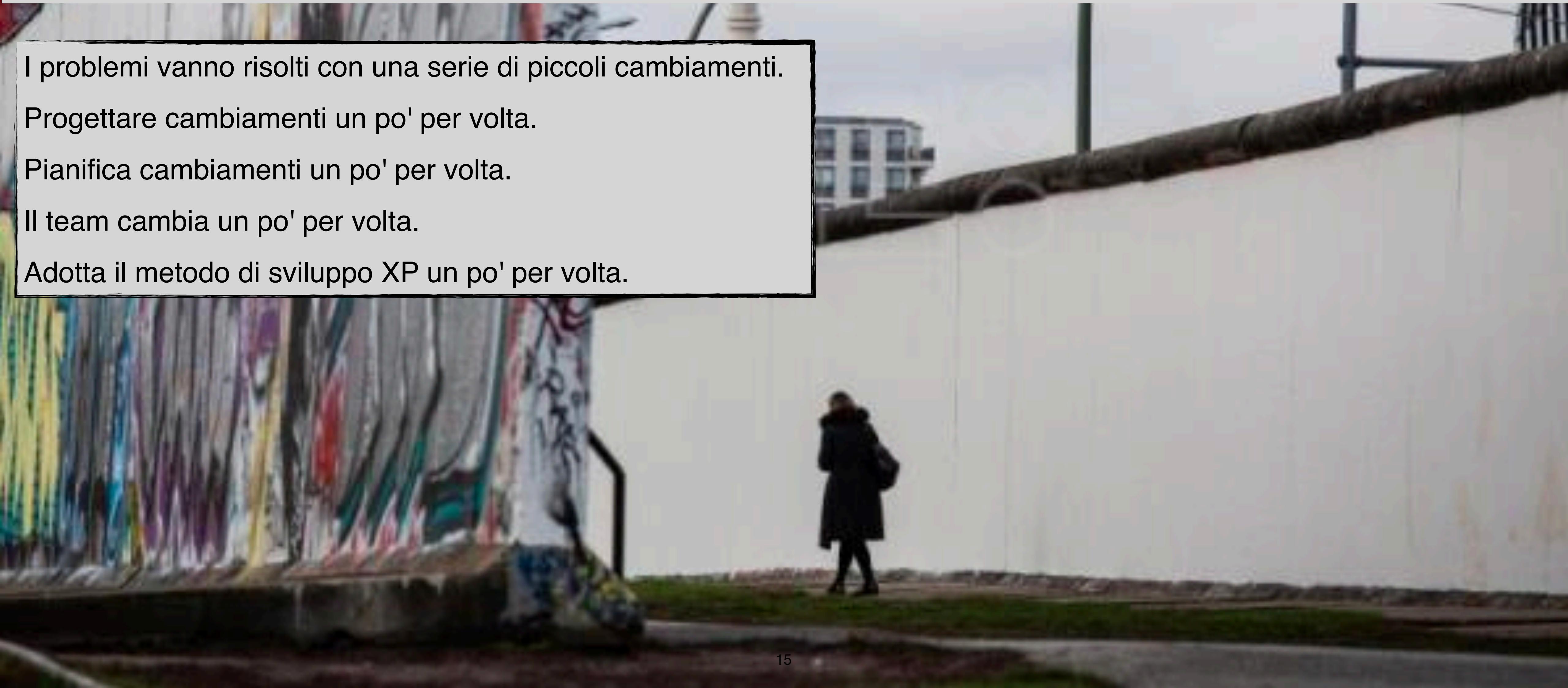
I problemi vanno risolti con una serie di piccoli cambiamenti.

Progettare cambiamenti un po' per volta.

Pianifica cambiamenti un po' per volta.

Il team cambia un po' per volta.

Adotta il metodo di sviluppo XP un po' per volta.



Cinque Principi di Base: Abbraccia il Cambiamento

Preserva la maggior parte delle opzioni mentre risolvi la maggior parte dei problemi pressanti.



Cinque Principi di Base: Qualità del Lavoro

Il team deve essere composto da persone che amano fare un buon lavoro.

Senza qualità:

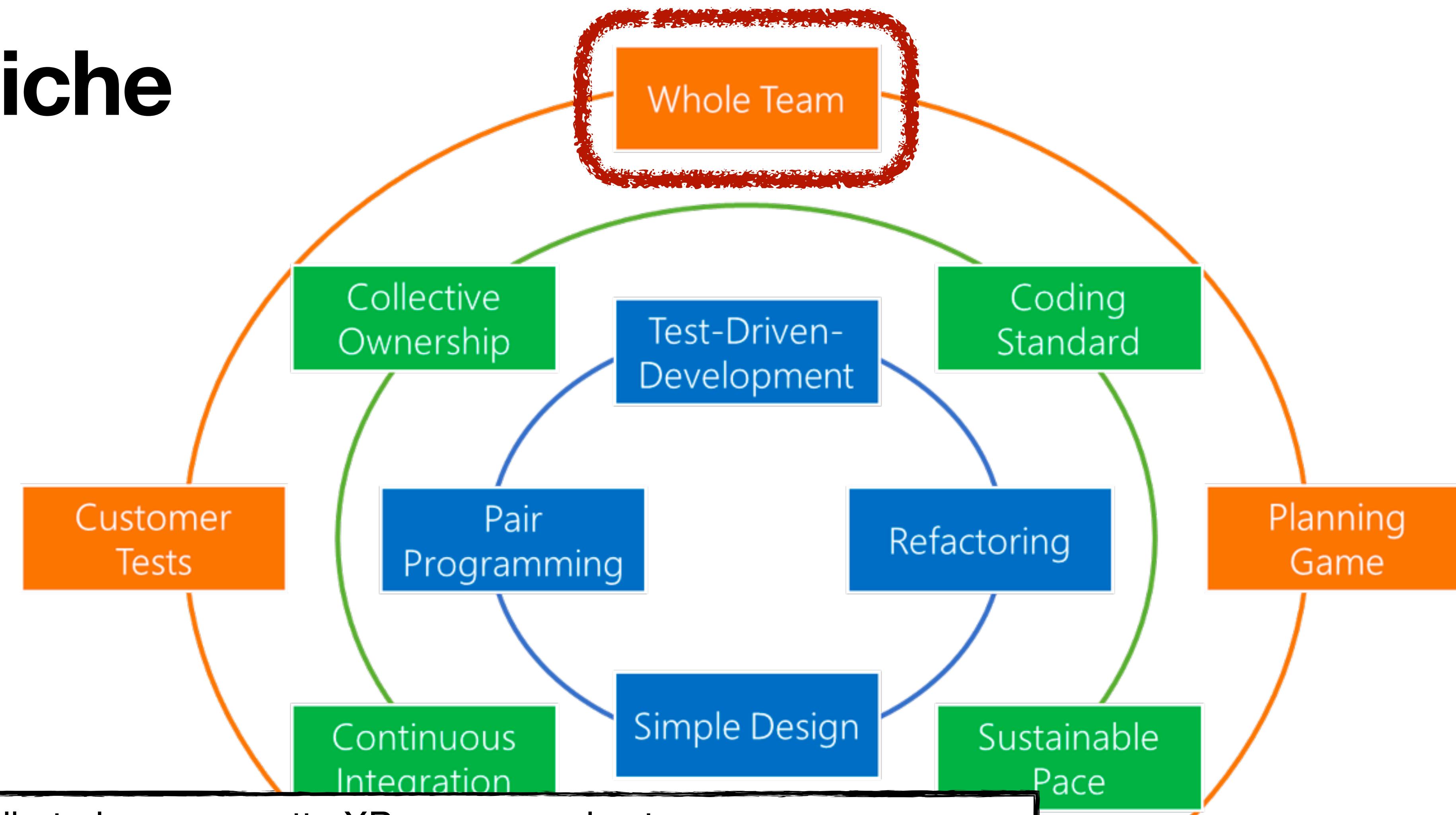
"you don't enjoy work";

"you don't work well";

il progetto va a rotoli.

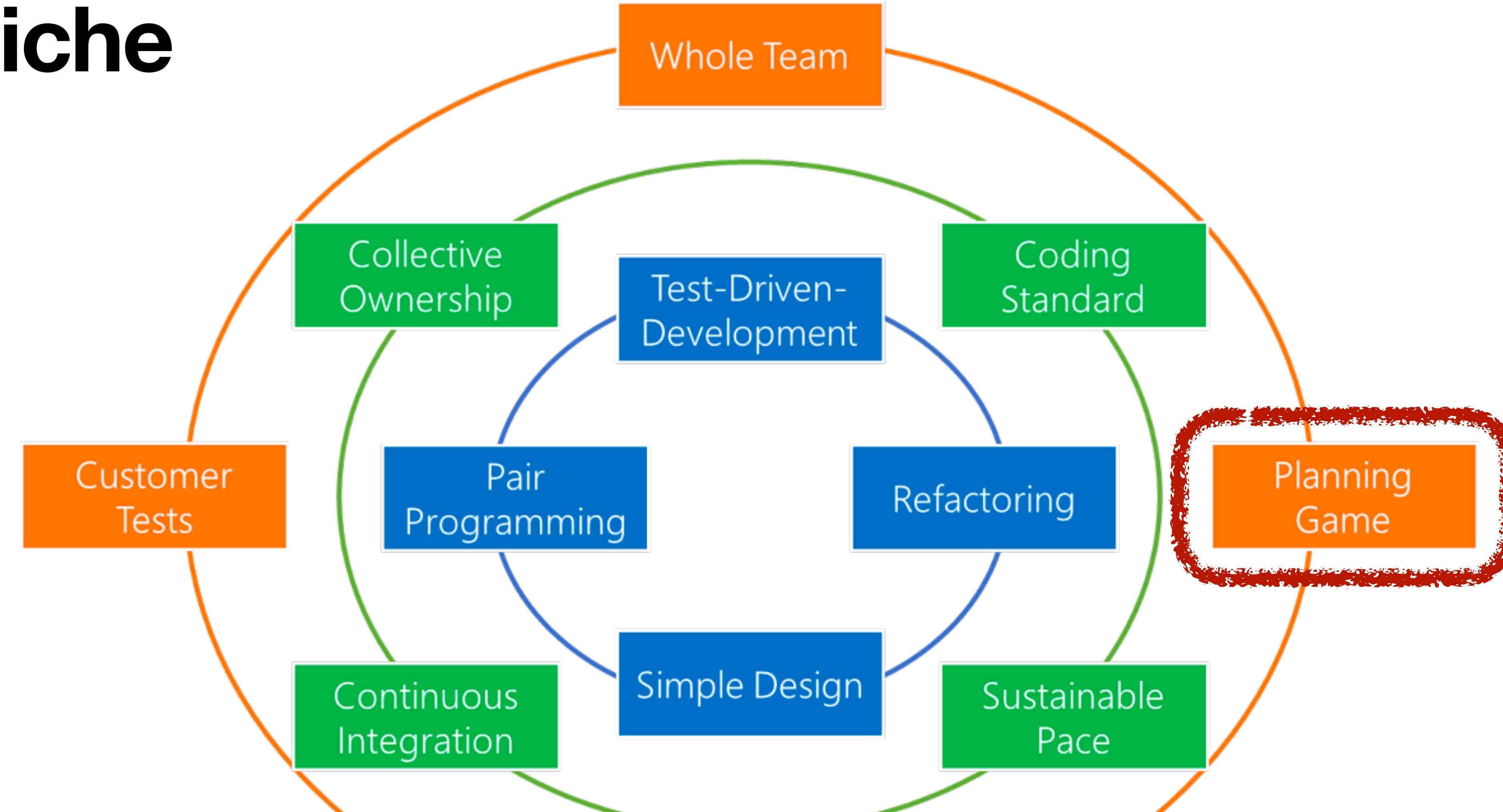


12 Pratiche



- Tutti i contributori a un progetto XP sono un unico team.
- Deve includere un rappresentante aziendale: il “Cliente”.
- I membri del team sono programmati, tester, analisti, coach, manager.
- I migliori team di XP non hanno specialisti.

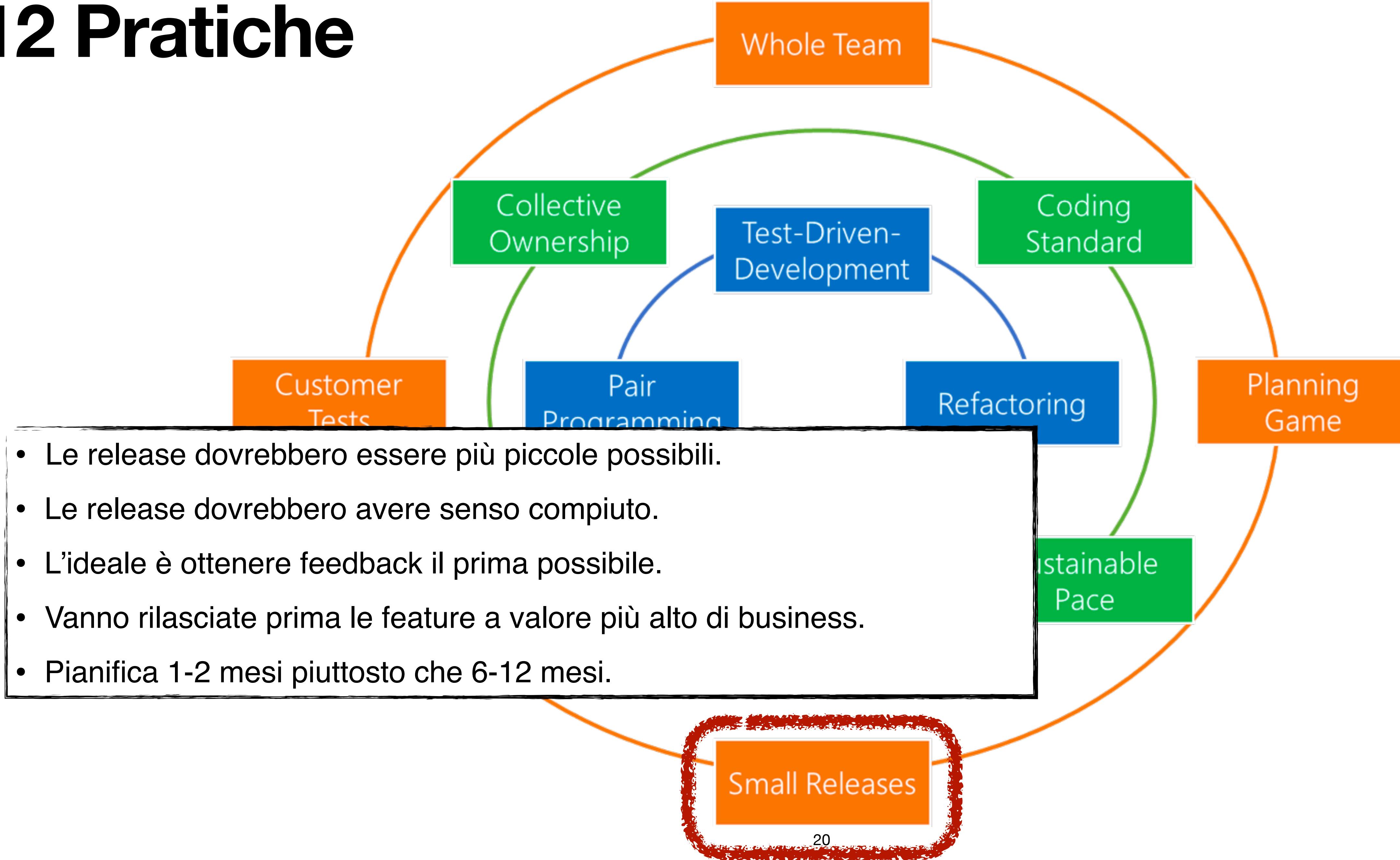
12 Pratiche



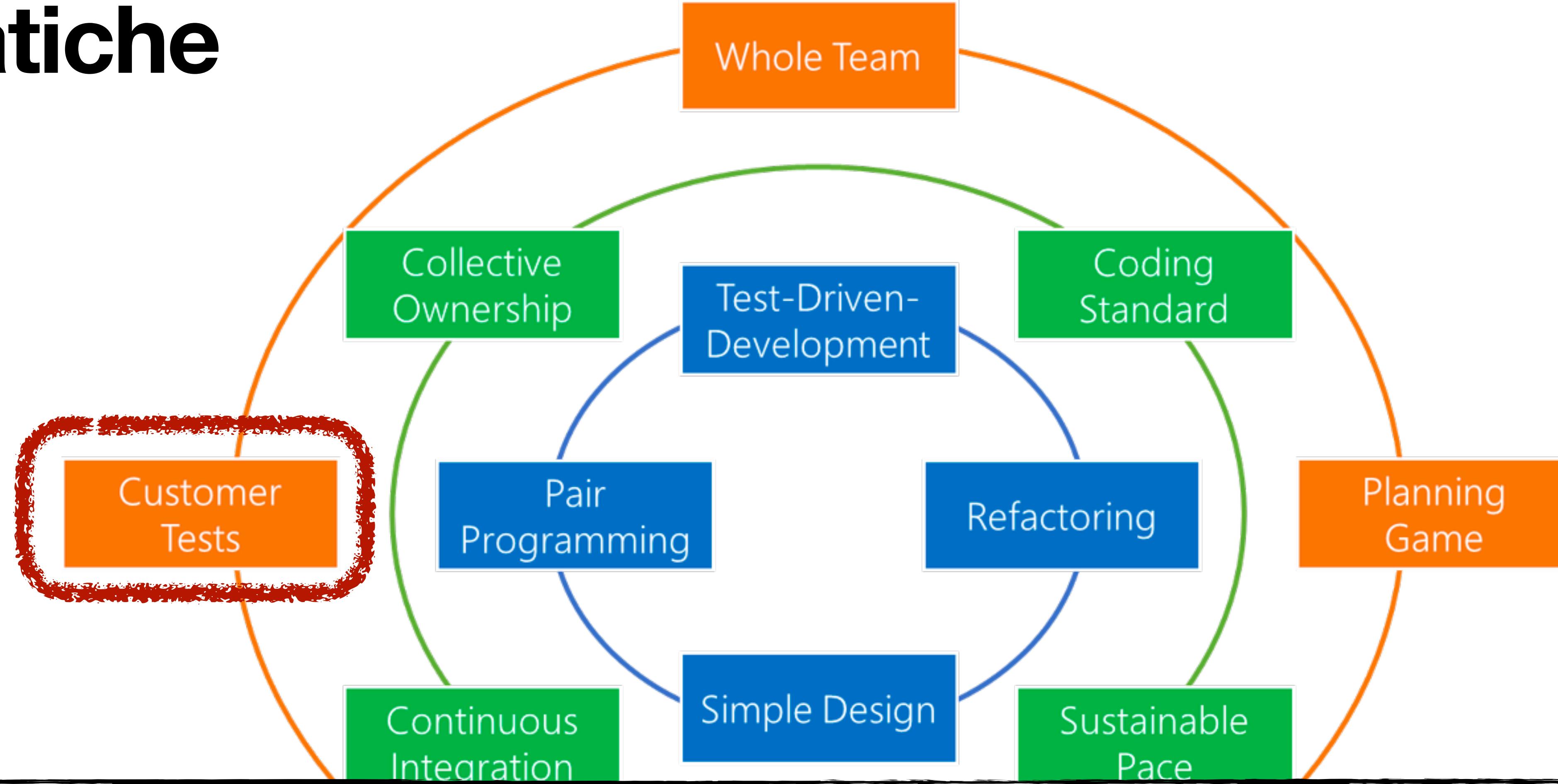
- I clienti scrivono story card e le prioritizzano.
- Gli sviluppatori stimano come implementare una user story.

Small Releases

12 Pratiche

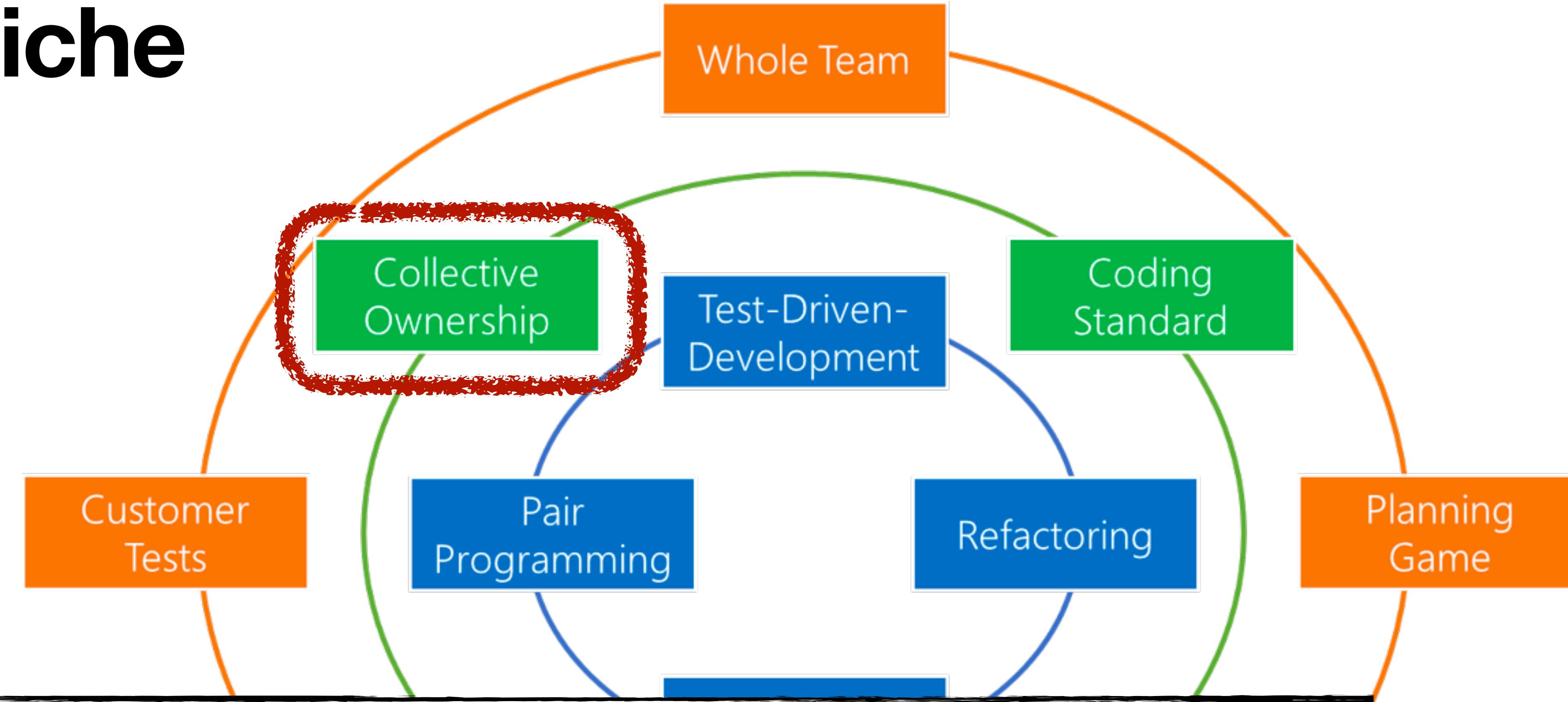


12 Pratiche



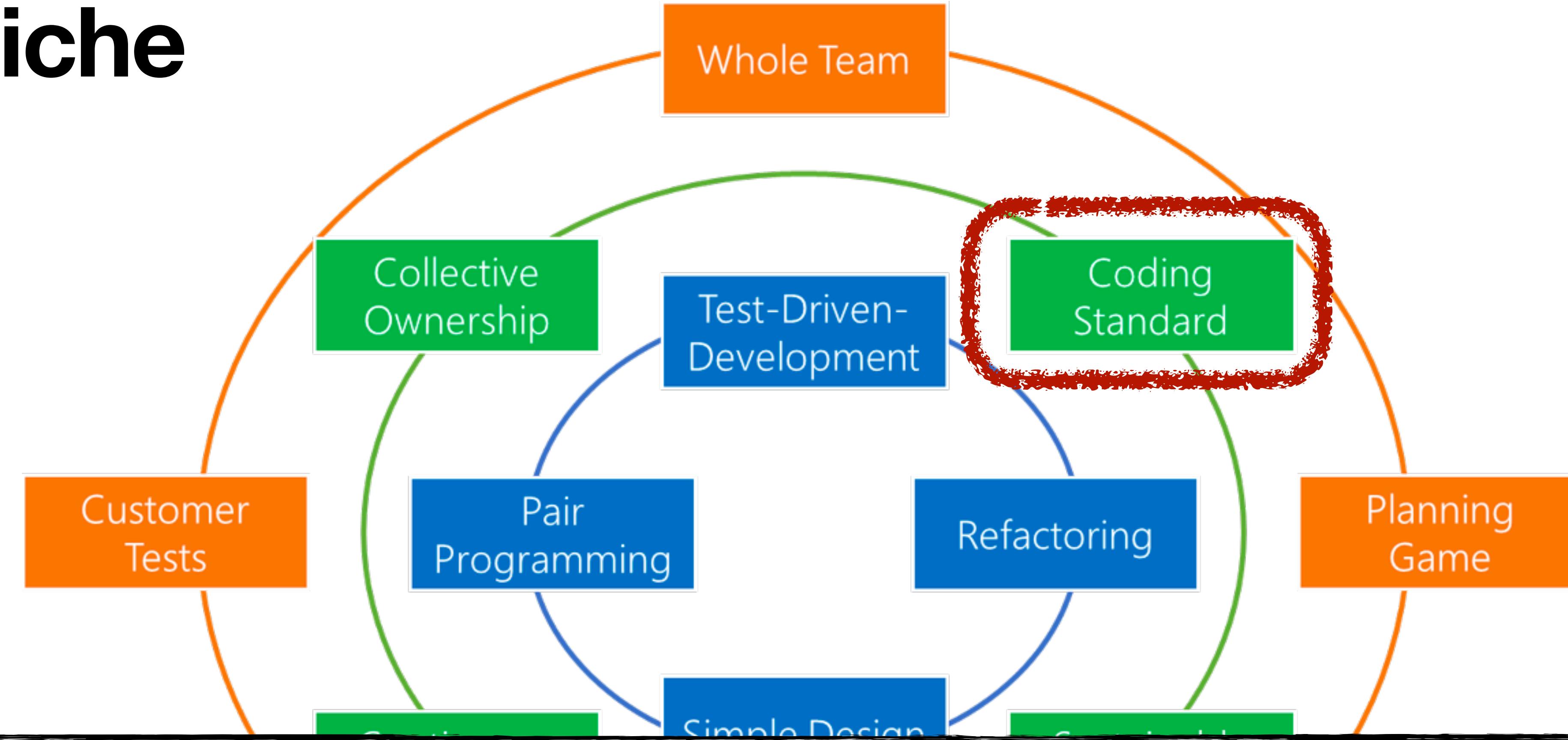
- Il Cliente definisce uno o più test di accettazione automatizzati per una funzionalità.
- Il team crea questi test per verificare che una funzionalità sia implementata correttamente.
- Una volta eseguito il test, il team garantisce che continui a funzionare correttamente da allora in poi.
- Il sistema migliora sempre, senza mai tornare indietro.

12 Pratiche



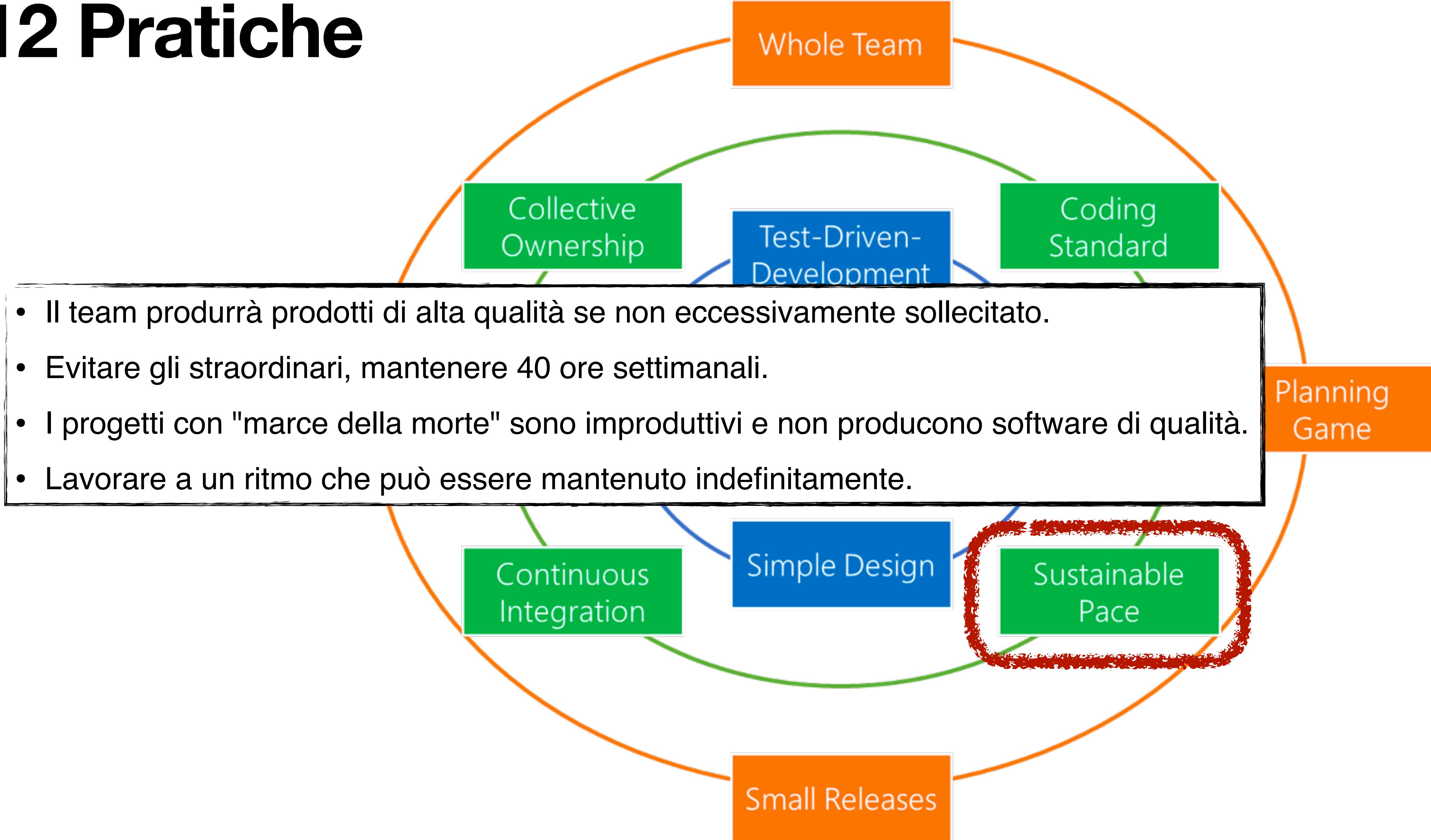
- Qualsiasi coppia di programmatori può migliorare qualsiasi codice in qualsiasi momento.
- Nessun "spazio di lavoro sicuro".
- Tutto il codice trae vantaggio dall'attenzione di molte persone.
- Evitare la duplicazione.
- I test del programmatore rilevano gli errori.
- Gli sviluppatori sono associati ad esperti quando lavorano su codice sconosciuto.

12 Pratiche

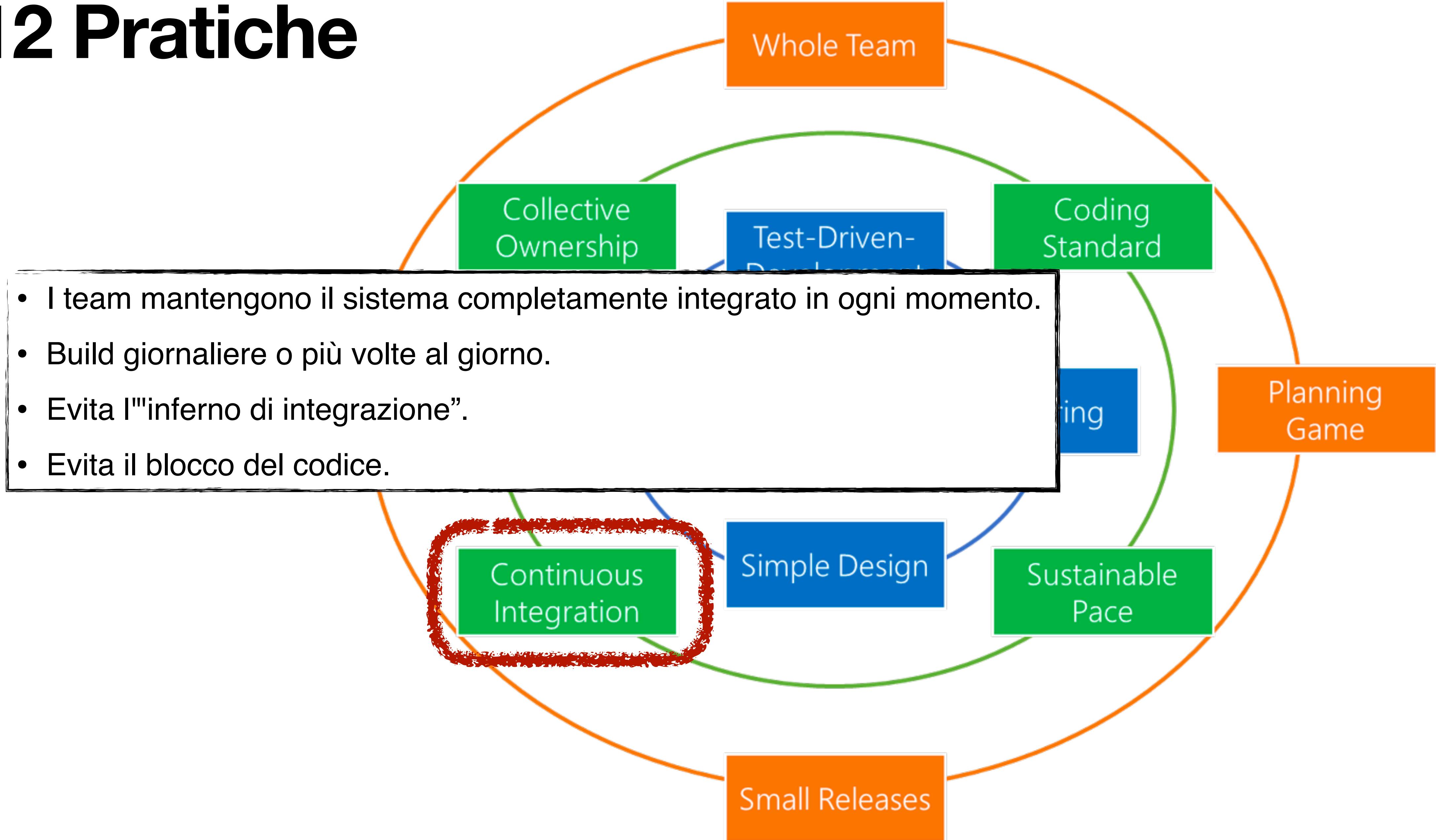


- Convenzioni per nomi e stili.
- Il team deve adottare una codifica standard.
- Cercare di rendere il proprio codice quanto più semplice possibile.
- Evitare di cambiare il codice in conseguenza a preferenze sintattiche dei singoli.
- Bisogna cercare di fare il minore lavoro possibile: il codice dovrebbe esistere una ed una sola volta.

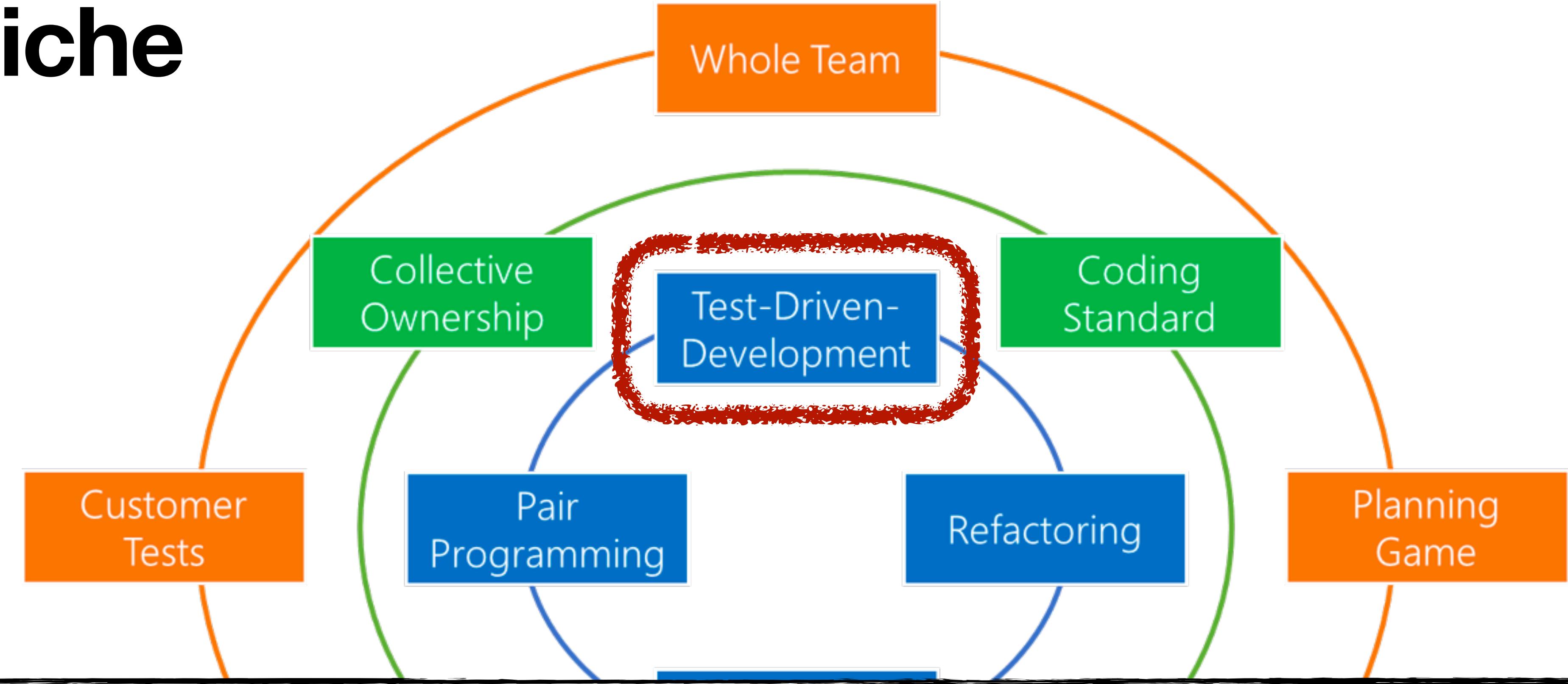
12 Pratiche



12 Pratiche



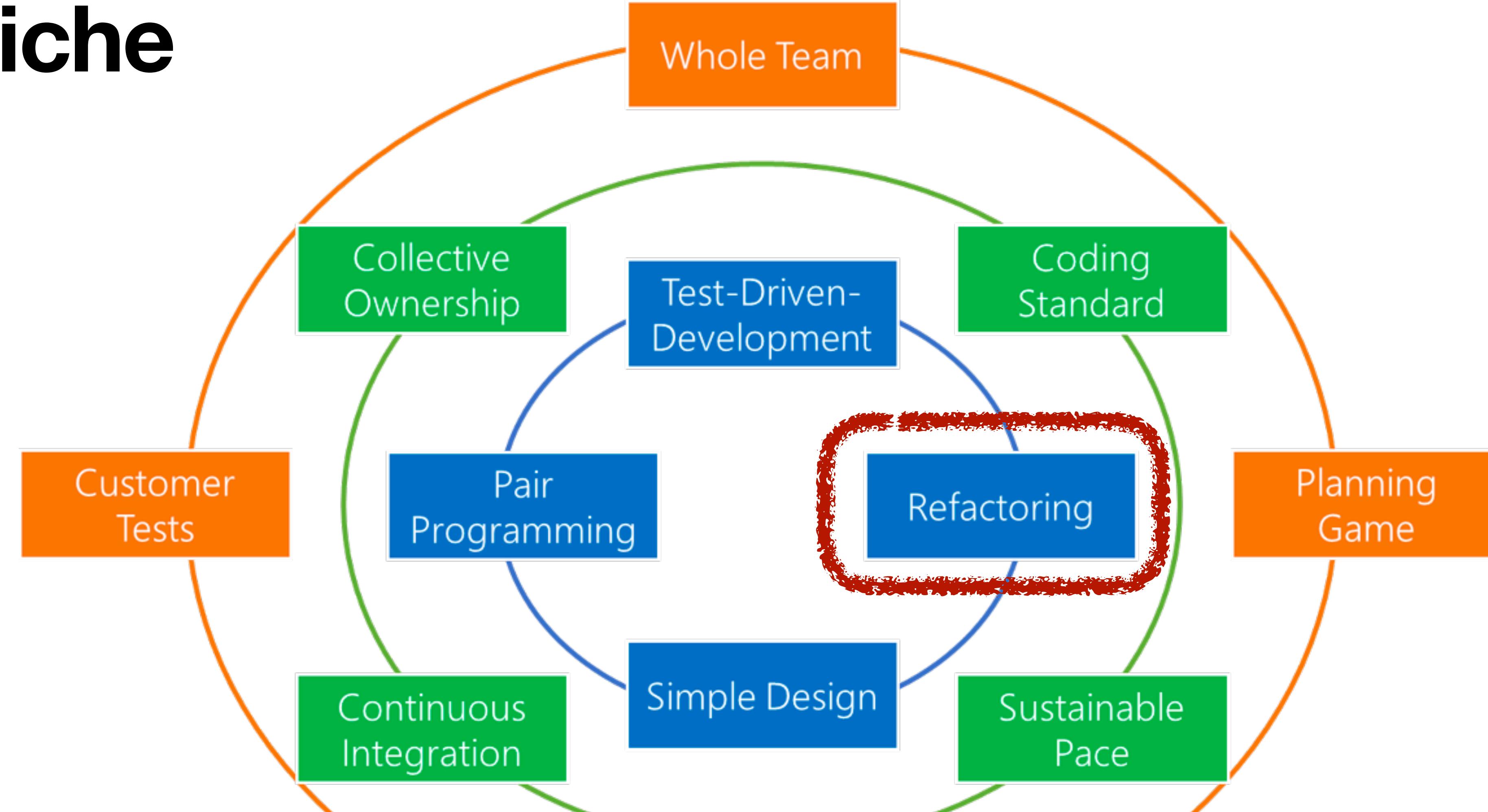
12 Pratiche



- I team sviluppando piccole test suite che testino il codice ancora da scrivere.
- Facile produrre codice con una copertura del test del 100 percento.
- Tutti i test sono raccolti insieme.
- Ogni volta che una coppia rilascia codice nel repository, ogni test deve essere eseguito correttamente.

Small Releases

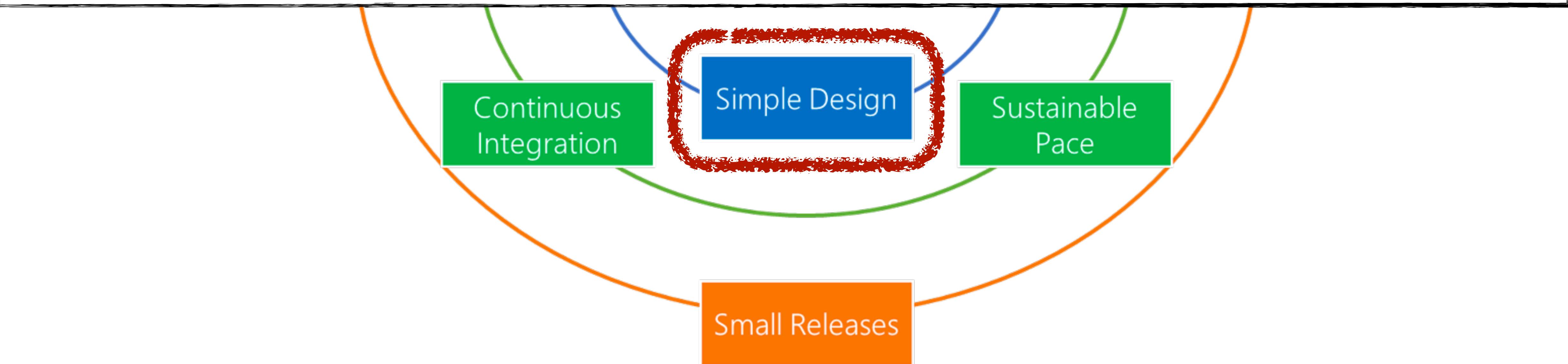
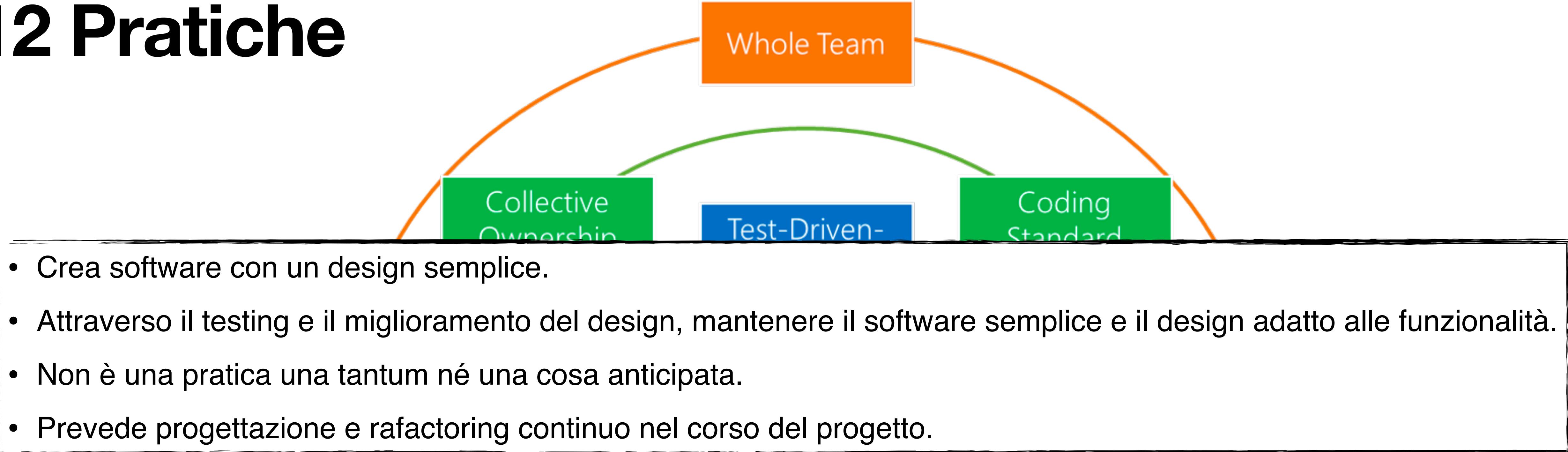
12 Pratiche



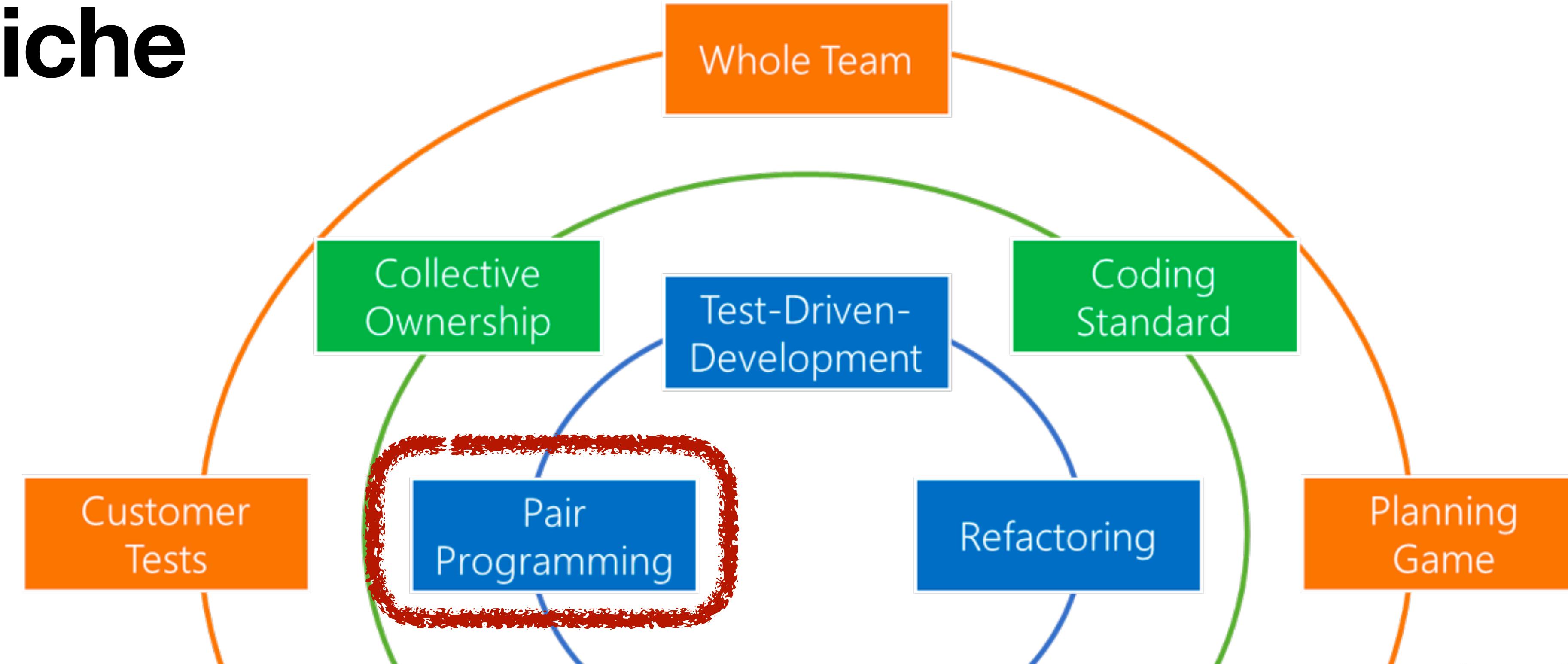
- Eliminare duplicazioni per aumentare la coesione e ridurre l'accoppiamento.
- Il refactoring è supportato dall'attività di testing.

Small Releases

12 Pratiche



12 Pratiche



- Tutto il software di produzione è costruito da due programmati, seduti fianco a fianco, con la stessa macchina.
- Tutto il codice di produzione viene quindi rivisto da almeno un altro programmatore.
- La ricerca mostra che il pair programming produce codice migliore rispetto allo sviluppo individuale.
- Il pair programming migliora la comunicazione nel team.

Small Releases

What's wrong with you?

Alcune critiche a XP Programming riportano che

- Non è realistico: incentrato sul programmatore, non orientato al business.
- Le specifiche sono dettagliate ma non scritte.
- La progettazione avviene dopo la fase di testing.
- Richiede refactoring costante.
- Richiede troppa disponibilità del cliente.
- Propone 12 pratiche troppo interdipendenti.