Le pratiche che stanno alla base dell'Extreme Programming sono in tutto 12 e si dividono in quattro aree.

La prima area riguarda i feedback a scala fine, che si possono ottenere mediante 4 pratiche:

1) Pair Programming: il codice deve essere prodotto da due persone operanti sullo stesso terminale. Un programmatore deve controllare il terminale e scrivere il codice dettagliatamente, l'altro deve pensare al progetto più ad alto livello per capire come semplificare il sistema e quali siano i casi di test da definire. I ruoli e le coppie vanno invertiti frequentemente. Questa pratica permette non di aumentare la funzionalità del codice rispetto alla programmazione individuale, ma di incrementarne la qualità, per poter risparmiare tempi e risorse nelle fasi successive del progetto.

2) Planning Game: il processo di pianificazione si svolge attraverso dei planning game, ovvero riunioni iterative dalla frequenza tipicamente settimanale che si dividono in due parti:

- Release Planning: consiste nel determinare i requisiti da includere nei prossimi rilasci e nel definire quando effettuare questi; questa fase coinvolge sia gli sviluppatori, sia i clienti, i quali determinano prima di tutto i requisiti più importanti da soddisfare nei rilasci e successivamente discutono con gli sviluppatori le funzionalità da includere, le date di rilascio ed eventuali modifiche ai requisiti.

- Iteration Planning: in questa fase i clienti non sono coinvolti, poiché si pianificano i task e le attività degli sviluppatori: i requisiti vengono tradotti in task diversi che vengono assegnati agli sviluppatori, stimandone i tempi necessari al completamento e confrontando al termine i risultati dei task con i requisiti iniziali.

3) Test Driven Development: prima di scrivere codice è necessario scrivere i test per capire da subito quali possono essere le condizioni di errore: è necessario, per ogni funzione ancora da implementare, scrivere uno unit test, per poi scrivere solamente il codice necessario a superare il test e successivamente svolgere refactoring del codice per aumentarne la qualità.

4) Whole team: i clienti, o chi per loro, sono coloro che useranno effettivamente il sistema, quindi un cliente o un suo rappresentante deve essere sempre disponibile per chiarimenti riguardo al progetto, per fornire sufficienti dettagli riguardo ai task degli sviluppatori, per risolvere incongruenze e decidere priorità anche attraverso i release planning. Ciò permette ai clienti di provare il sistema e di fornire feedback anticipatamente.

La seconda area consiste nel processo continuo, continuativo, basato su tre pratiche:

5) Continuous Integration: il team di sviluppo dovrebbe sempre operare sull'ultima versione del software per evitare ritardi successivi nello sviluppo dovuti a incoerenze o errori. Il ritmo di integrazione deve essere serrato: i programmatori dovrebbero integrare e aggiungere codice nelle repository ogni poche ore e mai prolungare cambiamenti al codice per più giorni: in questo modo si evita la scrittura divergente o frammentata di codice.

6) Design Improvement (refactoring): il design del sistema dovrebbe essere il più semplice possibile, quindi occorre ristrutturare il codice se si pensa di poterlo semplificare. Il refactoring è necessario nei casi in cui i cambiamenti funzionali necessitino di modifiche a copie diverse dello stesso codice o di codice simile, oppure una modifica in una parte del codice abbia conseguenze anche su molte altre. In questi casi è necessario modificare l'architettura del sistema, rendendolo più semplice e più generico.

7) Small Releases: La consegna del software deve essere fatta attraverso rilasci piccoli (i più piccoli possibili che contengano le funzionalità più importanti) e frequenti (un rilascio ogni 1 o 2 mesi) di funzionalità che

creino un valore concreto per il cliente. La dimensione ridotta dei rilasci "rassicura" il cliente sul progresso del progetto, anche al fine di mantenere il cliente attivo all'interno del team con consigli o critiche sulla base dell'utilizzo reale. La frequenza alta dei rilasci permette appunto di ottenere feedback significativi in tempo per eventuali cambiamenti importanti.

La terza area è quella della comprensione condivisa all'interno del team di sviluppo. Si basa su:

8) Coding Standards: Il codice scritto deve rispettare standard condivisi, al fine di mantenerne la consistenza e di renderlo più facile da leggere, modificare e ristrutturare per chi non l'ha scritto. è quindi opportuno anche scrivere codice il più possibile auto-documentato.

9) Collective Code Ownership: il codice non è di un programmatore in particolare, tutti gli sviluppatori devono essere responsabili per tutto il codice, quindi tutti possono modificare qualsiasi parte di esso. In questo modo è possibile accelerare lo sviluppo, poiché appena un programmatore trova un errore lo può correggere, anche se è possibile introdurre dei bug modificando una parte del codice mal compresa. Per ovviare a questo problema sono necessari unit test ben definiti.

10)Simple Design: come accennato, il progetto deve essere il più semplice possibile che soddisfi i requisiti, quindi occorre implementare solo quello che serve, nel momento in cui serve e senza aggiungere funzionalità ancora non previste. Il progetto deve essere ben testabile, comprensibile e navigabile. il livello di semplicità più alto si raggiunge con il refactoring. A prima vista si scontra con la programmazione per il riuso, in realtà il design va definito in modo da essere confidenti di saper aggiungere nuove funzionalità in futuro.

11)System Metaphor: per ogni progetto va definita una metafora, una storia che chiunque possa raccontare riguardo al funzionamento del sistema. Le metafore sono comprensibili anche ai clienti e forniscono convenzioni per i nomi da utilizzare nel codice, al fine di poterlo comprendere e riusare.

La quarta e ultima area, il benessere dei programmatori, è definita da un'unica pratica:

12) Sustainable Pace: gli sviluppatori non dovrebbero lavorare per più di 40 ore a settimana e non dovrebbero svolgere straordinari per più di due settimane consecutive. Persone riposate ovviamente possono operare in maniera più produttiva e creativa.