Usare Rational Unified Process per piccoli progetti:

Approfondimento di eXtreme Programming

Gary Pollice

White paper di Rational Software

TP 183, 3/01

Rational®
the software development company

Tabella dei contenuti

Sommario

Rational Unified Process« o RUP«è un framewordk completo del processo di sviluppo software corredato di numerose istanze pronte all'uso. I processi derivati da RUP variano da leggeri - rispondendo alle esigenze di progetti piccoli con cicli di produzione brevi - a processi più completi per rispondere ad esigenze di vasti ed eventualmente decentratiteam progettuali. Progetti di tutti i tipi e dimensioni hanno utilizzato RUP con successo. Questo white paper descrive come applicare RUP in modalità leggera per progetti piccoli.Viene, inoltre, descritto come applicare eXtreme Programming (XP) con efficacia all'interno del contesto più ampio di un progetto completo.

Introduzione

Breve storia

Una mattina un responsabile venne da me e mi chiese se potevo passare qualche settimana ad impostare un semplice sistema d'informazioni per una nuova venture che la compagnia stava per avviare. Ero stanco del mio progetto attuale ed ero eccitato all'idea di avviarne uno nuovo, perciò decisi di cogliere l'occasione - avrei potuto muovermi in modo rapido e sviluppare nuove grandi soluzioni, senza dover sottostare alla burocrazie ed alle procedure della grande organizzazione in cui lavoravo.

Le cose iniziarono alla grande. Per i primi 6 mesi ho lavorato sopratutto da solo. La mia produttività era incredibile ed ho svolto uno dei migliori lavori della mia carriera. I cicli di sviluppo erano veloci ed ero in grado, in genere, di produrre alcune nuove parti principali del sistema in poche settimane. Le interazioni con gli utenti erano semplici e dirette - eravamo tutti parte di un team compatto e potevamo esentarci da formalità e documentazioni. Anche nel caso della progettazione, tutto era molto semplice; il codice era la progettazione e viceversa. Andava tutto bene.

Per un pò. Al crescere del sistema, c'era sempre più lavoro da fare. Il codice esistente doveva evolversi con il mutare dei problemi e ci siamo trovati a rifinire le nozioni di cosa fosse necessario fare. Ho assunto diverse persone a supporto dello sviluppo. Abbiamo lavorato come una singola unità, spesso accoppiati su parti del problema. Ciò ha aumentato la comunicazione e così abbiamo potuto esentarci dalla formalità.

È passato un anno.

Abbiamo continuato ad aggiungere persone. Il team è cresciuto a tre, cinque, sette elementi. Ogni volta che iniziava una nuova persona, era necessaria una lunga curva di apprendimento e, senza il beneficio dell'esperienza, era sempre più difficile comprendere e spiegare l'intero sistema, anche come una semplice panoramica. Abbiamo iniziato ad acquisire i diagrammi su lavagna che mostravano la struttura generale del sistema e, in modo più dettagliato, le interfacce ed i concetti principali.

Abbiamo utilizzato ancora i test come principale mezzo per verificare che il sistema eseguisse ciò che doveva. Con tante nuove persone dal lato “utenti”, abbiamo pensato che i requisiti informali e le relazioni personali dei primi giorni del progetto non fossero più sufficienti. Ci volle più tempo per capire cosa dover costruire. come risultato, abbiamo conservato registrazioni scritte delle discussioni in modo da non dover continuamente ricordare cosa fosse stato deciso. Abbiamo inoltre capito che la descrizione dei requisiti e degli scenari d'uso erano di aiuto per educare i nuovi utenti al sistema.

Non appena il sistema iniziò a crescere di dimensione e complessità, avvenne qualcosa di inatteso - l'architettura del sistema richiedeva grande attenzione. Nei primi giorni l'architettura era in gran parte nella mia testa, più avanti su poche note o su lavagne a pagine mobili. Tuttavia, con l'aumentare delle persone al lavoro sul progetto diventava sempre più difficile mantenere l'architettura sotto controllo. Poiché non tutti disponevano della prospettiva storica come me, questi non erano capaci di vedere le implicazioni di una particolare modifica nell'architettura. Ci siamo trovati a dover definire i limiti strutturali sul sistema in termini più rigorosi. Tutte le modifiche che potevano aver avuto un effetto sulla struttura richiedeva il consenso del team e, in ultima analisi, la mia approvazione. Abbiamo scoperto tutto questo nel modo peggiore ed abbiamo dovuto imparare dure lezioni prima di ammettere a noi stessi che la struttura era davvero importante.

Questa è una storia vera. Descrive solo alcune delle dolorose esperienze di questo progetto. Le esperienze sono inusuali solo per un verso: molti di noi sono stati lì dall'inizio alla fine, per anni. Di solito le persone vanno e vengono da un progetto e non vedono l'impatto a valle delle loro azioni.

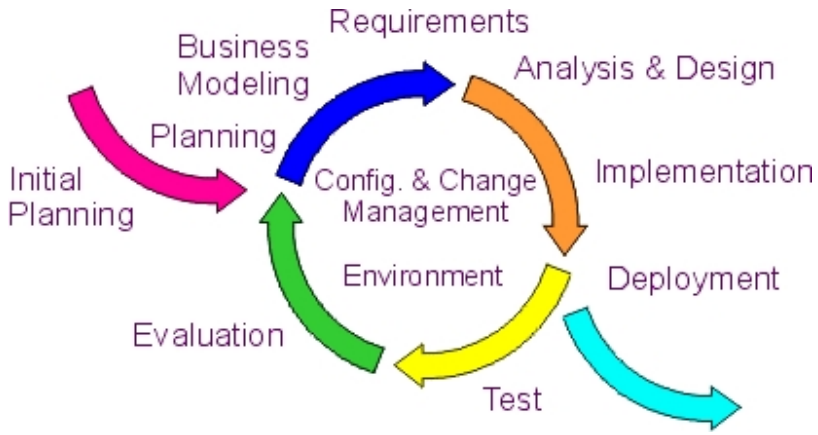
Questo progetto avrebbe potuto essere gestito con un pò più di processo. Troppo processo è eccessivo, ma la mancanza porta a nuovi rischi. Così come la persona che investe in azioni ad alto rischio vede solo alti ritorni, i gruppi che utilizzano troppo poco processo, ignorando i rischi maggiori nel loro ambiente di progetto, “sperano per il meglio ma sono impreparati al peggio.”

Panoramica

Il presente documento discute su come applicare il processo a progetti come quello appena descritto. L'attenzione è posta sul conseguimento del “giusto livello” del processo. Dalla comprensione delle sfide affrontate dal team di sviluppo e dall'ambiente business in cui opera, deriva il giusto livello di rigore del processo. Una volta comprese tali sfide, è necessario fornire solo abbastanza processo per mitigare i rischi. Non esiste una dimensione del processo che va bene per tutto, leggero o altro. Nelle successive sezioni, viene esplorata l'idea che il giusto livello del processo sia una funzione del rischio.

L'attenzione è posta su come ottenere il giusto livello di processo utilizzando due metodi popolari: Rational Unified Process o RUP ed eXtreme Programming (XP). Viene mostrato come personalizzare RUP ad un piccolo progetto e come esso investa aree non considerate da XP. La combinazione fornisce al team di progetto la guida necessaria per diminuire i rischi e conseguire l'obiettivo di creare un prodotto software.

RUP è un framework di processo sviluppato da Rational Software. È una metodologia di sviluppo iterativo basata sulle sei procedure ottimali verificate dall'industria (vedere l'appendice di RUP). Nel tempo, un progetto basato su RUP attraversa quattro fasi: inizio, elaborazione, costruzione e transizione, Ogni fase contiene una o più iterazioni. In ogni iterazione viene investito un impegno in diverse quantità per ciascuna delle diverse discipline (o flussi di lavoro) come i requisiti, l'analisi e la progettazione, il test e così via. L'indicazione principale per RUP è la diminuzione dei rischi. RUP è stato rifinito per l'utilizzo in migliaia di progetti con migliaia di clienti e partner di Rational. Il seguente diagramma illustra il flusso attraverso un'iterazione tipica:



Flusso d'iterazione tipico

Come esempio di come i rischi possano influenzare un processo, è possibile chiedersi se modellare il business. Nel caso di rischi significativi che non compresi dal business comportano la costruzione di un sistema fallato, è probabilmente necessario eseguire un qualche tipo di modellazione. È necessario essere rigorosi sull'impegno nella modellazione? Dipende dal pubblico - se un piccolo team utilizza il risultato in modo informale, è necessario creare solo note informali. Ma se altri nell'organizzazione utilizzeranno o revisioneranno tali risultati, probabilmente è necessario un impegno maggiore, rivolto verso una presentazione corretta e comprensibile.

È possibile personalizzare RUP per soddisfare le necessità di quasi ogni progetto. Se nessun dei percorsi o dei processi pronti all'uso si adatta alle proprie specifiche necessità, è facilmente possibile produrne uno proprio. Un percorso descrive come il progetto intende utilizzare il processo edunque rappresenta una specifica istanza di processo per quel progetto. Questo significa che RUP può essere leggero o pesante secondo le necessità illustrate nel presente documento.

XP è un leggero processo incentrato sul codice per piccoli progetti (vedere l'appendice di XP). Nasce da un'invenzione di Kent Beck ed è salito alla ribalta di fronte alle industrie software per il progetto dell'elenco dipendenti di C3 alla Chrysler Corporation nel 1997. Come RUP si basa su iterazioni che incorporano diverse procedure come piccoli rilasci, semplici progettazioni e continue integrazioni. XP promuove diverse tecniche efficaci per appropriati progetti e circostanze; tuttavia, esistono ruoli, attività ed assunzioni nascoste.

RUP e XP provengono da diverse filosofie. RUP è un framework di componenti, metodi e tecniche di processo che è possibile applicare ad ogni specifico progetto software; ci si aspetta che l'utente specializzi il RUP. XP, d'altra parte, è un

processo più vincolato che necessita di aggiunte per poter soddisfare un progetto di sviluppo completo. Queste differenze spiegano la percezione nella comunità di sviluppo software generale: per i grandi sistemi RUP è la risposta ai problemi; la comunità di un piccolo sistema invece preferisce XP. L'esperienza indica che la maggior parte dei progetti software si trova nel mezzo - nella ricerca del conseguimento del giusto livello di processo per la loro situazione. Né è sufficiente per loro la fine dello spettro.

Combinando la grandezza di RUP con alcune delle tecniche di XP è possibile ottenere la giusta quantità di processo che mette d'accordo tutti i membri di un progetto e che affronta tutti i principali rischi del progetto. Per un team al lavoro su un piccolo progetto in un ambiente dove vi è grande fiducia e in cui l'utente è parte integrale del team, in questo caso XP può operare molto bene. Con un team maggiormente distribuito e con la crescita della base del codice o se l'architettura non è ancora ben definita, è necessario qualcos'altro. È necessario più di XP per progetti che hanno uno stile “contrattuale” dell'interazione dell'utente. RUP è un framework con cui è possibile estendere XP con un insieme più robusto di tecniche, in caso di necessità.

La parte restante del presente documento descrive un piccolo processo basato sulle 4 fasi di RUP. In ognuna, vengono identificate le attività e gli artefatti prodotti.[1] Anche se RUP e XP identificano ruoli e responsabilità differenti, queste non vengono trattate in questa sede. Per ogni organizzazione o progetto, tutti gli attuali membri devono essere associati ai propri ruoli.

Avvio progetto - Inizio

L'inizio è importante per i nuovi impegni dello sviluppo, in cui è necessario affrontare importanti rischi business e di requisiti prima di portare avanti il progetto. Per i progetti concentrati sui miglioramenti di un sistema esistente, la fase iniziale è più breve ma è sempre incentrata sull'assicurare che il progetto sia possibile da fare e che valga la pena di farlo.

Durante la fase di inizio, viene creato il caso business per la costruzione del software. La Visione è un artefatto chiave prodotto durante questa fase. È una descrizione di alto livello del sistema. Spiega cos'è ed anche chi lo usa, perché viene usato, quali caratteristiche deve avere e quali vincoli esistono. La Visione può essere molto breve, anche solo un paragrafo o due. Spesso la Visione contiene le caratteristiche principali che il software deve fornire al cliente.

Il seguente esempio mostra una Visionemolto breve scritta per il progetto per ri-programmare il sito web esterno di Rational.

> Per guidare Rational alla posizione di leader mondiale nell'e-development (tool, servizi e le procedure migliori), rinforzando le relazioni con il cliente attraverso una presenza dinamica e personalizzata nel web che fornisce contenuti concepiti per il visitatore, self-service e supporto. Il nuovo processo e l'abilitazione delle tecnologie dà forza ai provider dei contenuti per velocizzarne la pubblicazione e la qualità attraverso una soluzione automatizzata e semplice.

Quattro attività essenziali in questa fase e descritte in RUP sono:

- formulare l'ambito del progetto — Se deve essere prodotto un sistema, è necessario sapere cos'è e come possa soddisfare gli stakeholder. In questa attività vengono acquisiti il contesto ed i più importanti requisiti con sufficiente dettaglio per desumere i criteri di accettazione del prodotto.
- pianificare e preparare il caso business — Con la Visione a fungere da guida, definire la strategia di diminuzione dei rischi, sviluppare un piano di progetto iniziale e identificare costi, pianificazioni e bilanciamenti di profitto.
- sintetizzare una struttura potenziale — Se il sistema sotto analisi possiede pochi elementi di novità ed ha una struttura ben conosciuta, è possibile saltare questo passo. Appena è noto quanto richiesto dal cliente, viene impiegato del tempo per indagare sulle strutture potenziali. La nuova tecnologia porta con sé il potenziale per nuove e migliori soluzioni per i problemi software. Investire tempo nel processo per valutare l'acquisto rispetto ai bilanciamenti della costruzione, così come per la selezione di tecnologie e magari per lo sviluppo di un prototipo iniziale, può ridurre alcuni dei principali rischi per il progetto.
- preparare l'ambiente di progetto — Ogni progetto necessita di un ambiente. Sia utilizzando alcune delle tecniche di XP come la programmazione accoppiata, che altre più tradizionali, resta necessario determinare risorse fisiche, tool di software e procedure che il team deve seguire.

[1] XP definisce tre fasi: esplorazione, impegno e guida. Non si adattano bene alle fasi di RUP perciò in genere vengono scelte le quattro fasi di quest'ultimo per a descrizione del processo.

Non ci vuole molto “tempo di processo” per eseguire la fase d'inizio su un piccolo progetto. Spesso è possibile concluderla in un paio di giorni o meno. Le seguenti sezioni descrivono gli artefatti previsti, diversi dalla Visione, per questa fase.

Un caso business approvato

Gli stakeholder hanno la possibilità di accordarsi sul fatto che, da una prospettiva business, valga la pena portare avanti il progetto. RUP e XP concordano che è meglio scoprire prima se ne vale la pena piuttosto che spendere risorse su un progetto condannato. XP, come descritto in Planning Extreme Programming[1], è poco chiaro in merito a come i progetti debbano diventare e quali ruoli debbano essere coinvolti (sembra più chiaro nel contesto di un business o sistema esistente), ma nella fase di esplorazione XP si accorda con gli equivalenti artefatti della fase di inizio di RUP.

Se le questioni business vengono considerate in modo informale come in XP o se, invece, il caso business viene reso un artefatto di progetto di prima classe, come in RUP, è necessario considerarle.

Elenco dei rischi

Mantenere l'elenco rischi lungo tutto il progetto. Può essere un semplice elenco con strategie di diminuzione pianificate. I rischi vengono elencati per importanza. Tutti coloro che sono associati al progetto possono vedere quali sono i rischi e come poterli affrontare in un momento qualsiasi del tempo. Kent Beck descrive una serie di rischi che XP affronta ed il modo in cui li affronta, ma non viene fornito nessun approccio generale per la loro gestione.[2]

Piano di progetto preliminare

Valutazioni di risorsa, ambito e piani di fase sono inclusi in questo piano. Su ogni progetto, queste valutazioni cambiano continuamente ed è necessario tenerle sotto controllo.

Piano di accettazione progetto

Avere un piano rigoroso o meno dipende dal tipo di progetto. È necessario decidere in che modo il cliente valuta il successo del progetto. Per un progetto di XP, il cliente crea test di accettazione. In un processo più generale, il cliente non può costruire test, ma deve condurre i criteri di accettazione direttamente o mediante un altro ruolo, come l'analista di sistema, che interagisce direttamente con lui. Possono esserci altri criteri di accettazione come la produzione della guida e della documentazione dell'utente finale, che però XP non tratta.

Un piano per l'iterazione di elaborazione iniziale

In un progetto basato su RUP, viene pianificata in dettaglio ogni iterazione alla fine di quella precedente. Viene valutato il progresso rispetto ai criteri stabiliti all'inizio dell'iterazione. XP fornisce alcune buone tecniche per il controllo e la misurazione del successo di un'iterazione. Le metriche sono semplici ed è possibile incorporarle facilmente nel piano d'iterazione e nei criteri di valutazione.

Elaborazione

L'obiettivo della fase di elaborazione consiste nel creare la linea di base dell'architettura del sistema per fornire una base stabile per le dimensioni dello sforzo di progettazione e di implementazione nella fase di costruzione. L'architettura si sviluppa dalla considerazione dei requisiti più importanti (quelli che hanno un impatto forte sull'architettura del sistema) e da una valutazione del rischio. La stabilità dell'architettura viene valutata attraverso uno o più prototipi strutturali.

In RUP, le attività di progettazione vertono sulla nozione di architettura disistema e, per i sistemi software intensivi, su quella di architettura software. L'utilizzo delle architetture di componente è una delle sei procedure ottimali di sviluppo software appartenenti a RUP, che consiglia

[1] Questo è uno dei tre libri attualmente pubblicati su eXtreme Programming.

[2] In eXtreme Programming eXplained, Kent Beck descrive otto rischi e come vengono affrontati da XP (pp. 3-5). Si invita il lettore a guardarli e determinare se sono sufficienti. È plausibile pensare che vi siano molti altri rischi e che una strategia di gestione dei rischi generale sia una parte necessaria per ogni processo.

di investire tempo sviluppando e mantenendo l'architettura. Il tempo speso per quest'impegno mitiga i rischi associati ad un sistema fragile e non flessibile.

XP Riprende la nozione di architettura per “metafora.” La metafora cattura parte dell'architettura, mentre il resto si evolve come naturale risultato dello sviluppo del codice. XP assume che l'architettura emerga dalla produzione della progettazione più semplice e dal continuo refactoring al codice.

In RUP, l'architettura è più di una metafora. Durante l'elaborazione vengono costruite architetture eseguibili, da cui è possibile ridurre molti dei rischi associati all'incontro di requisiti non funzionali come le prestazioni, l'affidabilità e la robustezza. Nella letteratura di XP, è possibile desumere che alcune cose prescritte da RUP per la fase di elaborazione, ossia nello specifico una concentrazione non necessaria su ciò che XP chiama infrastruttura, rappresentano una perdita di tempo. XP afferma che l'impegno speso nella costruzione dell'infrastruttura in anticipo rispetto alla necessità di averla, conduce a soluzioni eccessivamente complesse ed alla produzione di cose che non hanno valore per il cliente. In RUP, struttura ed infrastruttura non sono la stessa cosa.

L'approccio all'architettura è diverso tra RUP e XP. RUP suggerisce di porre attenzione all'architettura per impedire i rischi associati all'aumento dell'ambito nel tempo, della dimensione del progetto e delle nuove tecnologie. XP assume un'architettura esistente o che l'architettura è semplice abbastanza o ben conosciuta al punto che può evolvere mentre viene codificata. XP consiglia di non progettare per il domani, ma di implementare oggi. La convinzione è che il domani si prende cura da sé se la progettazione viene mantenuta il più semplice possibile. RUP invita a valutare i rischi di tale proposizione. Se il sistema o parte di esso va riscritto in futuro, XP indica che ciò è ancora meglio nonché spesso meno dispendioso della pianificazione per la possibilità oggi. Per alcuni sistemi questo è vero e con RUP la considerazione del rischio durante la fase di elaborazione conduce a questa conclusione. RUP non impone questa verità per tutti i sistemi e l'esperienza suggerisce che nel caso di sistemi più grandi, complessi e nuovi, può essere disastrosa.

Se è vero che investire troppa attenzione per possibilità che potrebbero non avvenire mai è inutile, porre la giusta dose di attenzione al futuro resta una cosa prudente da fare. Quante compagnie possono permettersi di riscrivere continuamente o persino di eseguire il refactoring del codice?

Per ogni progetto, è necessario eseguire almeno le seguenti tre attività durante l'elaborazione:

- Definire, convalidare e creare una linea di base per l'architettura — Utilizzare l'elenco rischi per sviluppare l'architettura candidata a partire dalla fase d'inizio. L'interesse è nell'assicurare che il software immaginato sia possibile. Nel caso di poca novità nella tecnologia scelta o di poca complessità del sistema, questo compito non comporta molto tempo. Se si sta effettuando un'aggiunta ad un sistema esistente, il compito può non essere necessario se non lo sono le modifiche all'architettura. Nel caso di reali rischi strutturali presenti, non bisogna consentire la modifica dell'architettura.

  Come parte di questa attività, è possibile eseguire alcune selezioni di componenti e prendere decisioni di acquisto/costruzione/riutilizzo. Se ciò richiede molto impegno, è possibile svolgerla come attività separata.

- Perfezionamento della Visione — Durante la fase d'inizio, è stata sviluppata una Visione. Nel momento in cui viene determinata la possibilità del progetto e gli stakeholder hanno il tempo per revisionare e commentare il sistema, possono essere apportate modifiche al documento ed ai requisiti di Visione. Tali revisioni avvengono in genere durante l'elaborazione. Alla fine di questa fase, è stata stabilita una solida comprensione dei casi d'uso più importanti che guidano le decisioni strutturali e di pianificazione. Gli stakeholder devono convenire che la visione attuale può verificarsi se viene eseguito l'attuale piano per sviluppare il sistema completo, nel contesto dell'attuale architettura. La quantità di modifica deve diminuire durante le iterazioni successive, ma è consigliabile destinare un pò di tempo in ogni iterazione per la gestione dei requisiti.

- Creare e definire una linea di base per piani d'iterazione della fase di costruzione— Inserire adesso i dettagli del piano. Alla fine di ogni iterazione di costruzione, rivedere i piani e correggerli se necessario. Le correzioni in genere sono necessarie poiché l'impegno è stato valutato in modo non corretto, l'ambiente business o i requisiti sono cambiati. Assegnare priorità ai casi d'uso, agli scenari ed agli impegni tecnici e poi assegnarli alle iterazioni. Alla fine di ogni iterazione bisogna pianificare di avere un prodotto funzionante che dia valore agli stakeholder.

È possibile eseguire altre attività durante l'elaborazione. Si consiglia di stabilire l'ambiente di test e di avviarne lo sviluppo. È possibile che non ci sia un codice dettagliato, ma è comunque possibile progettare e probabilmente implementare test d'integrazione. I programmatori devono essere pronti a sviluppare test di unità e capire come utilizzare i tool di test selezionati per il progetto. XP consiglia di scrivere il test prima del codice. È una buona idea, specie se questo va aggiunto ad un codice esistente. In qualunque modo si scelga di eseguire il test, il tempo per stabilire un regime di test regolare è nell'elaborazione.

La fase di elaborazione descritta da RUP contiene elementi delle fasi di esplorazione e di impegno di XP. L'approccio di XP ai rischi tecnici, come la novità e la complessità, è la soluzione “spike” , consistente ad esempio nel prendere tempo per eseguire esperimenti per valutare l'impegno. Questa tecnica è efficace in molti casi, con un rischio più grande non racchiuso in un singolo caso d'uso o storia, è necessario applicare una maggiore quantità di impegno per assicurare l'esito positivo del sistema ed un'accurata stima dell'impegno stesso.

Durante l'elaborazione vengono generalmente aggiornati artefatti come i requisiti e l'elenco rischi dalla fase di inizio. Gli artefatti che possono apparire durante l'elaborazione sono:

- SAD (Software Architecture Document) — SAD è un artefatto composito che fornisce una singola fonte di informazioni tecniche nel corso del progetto. Alla fine dell'elaborazione può contenere dettagliate descrizioni sui casi d'uso strutturalmente più importanti o l'identificazione di meccanismi chiave ed elementi della progettazione. Se il progetto rafforza un sistema esistente, è possibile utilizzare un SAD precedente oppure ritenere che non ci sia quasi rischio nel non utilizzarlo. In tutti i casi, è necessario eseguire i processi pensati che conducono al documento.
- Piani d'iterazione per la costruzione — Pianificare il numero di iterazioni di costruzione durante l'elaborazione. Ogni iterazione ha specifici casi d'uso, scenari ed altri elementi di lavoro ad esso associato. Queste informazioni vengono catturate e inserite in una linea di base nei piani d'iterazione. Revisionare ed approvare i piani come parte dei criteri d'uscita per l'elaborazione.

Per progetti molto brevi e piccoli è possibile unire l'iterazione di elaborazione con l'inizio e la costruzione. Le attività essenziali vengono ancora eseguite, ma si riducono le risorse per la pianificazione e le revisioni.

Modello di caso d'uso iniziale

Per quanto possa apparire formale e intimidatorio, è chiaro e diretto. I casi d'uso corrispondono alle “storie” scritte dal cliente in XP. La differenza è che un caso d'uso è una serie completa di azioni iniziata da un attore, qualcuno o qualcosa al di fuori del sistema che fornisce valore visibile. Il caso d'uso può contenere diverse storie di XP. Per definire l'ambito del progetto, RUP raccomanda di identificare casi d'uso e attori durante la fase di inizio. Porre l'attenzione su una serie completa di azioni dal punto di vista dell'utente aiuta nella divisione del sistema in più parti che forniscono valore. Questo aiuta a determinare le caratteristiche d'implementazione appropriate in modo da avere qualcosa da consegnare al cliente alla fine di ogni iterazione (tranne possibilmente le prime iterazioni, di inizio e di elaborazione).

Sia RUP che XP aiutano ad assicurare che non ci si trovi mai nella posizione di avere l'80% del sistema completo, senza avere però nulla in forma distribuibile. È sempre desiderabile il poter rilasciare il sistema per fornire un qualche valore al cliente.

Il modello di caso d'uso, a questo punto, identifica casi d'uso e attori senza quasi alcun dettaglio a supporto. Può trattarsi semplicemente di diagrammi di testo o UML (Unified Modeling Language) fatti a mano o con tool di disegno. Questo modello aiuta ad assicurare di aver incluso le giuste caratteristiche per gli stakeholder e di non aver dimenticato nulla, oltre a consentire di vedere facilmente l'intero sistema. I casi d'uso vengono messi in ordine di priorità in base a diversi fattori come i rischi, l'importanza per il cliente e le difficoltà tecniche.

Nessuno degli artefatti della fase d'inizio deve essere eccessivamente rigoroso o grande. Renderli semplici o rigorosi secondo necessità. XP contiene una guida sulla pianificazione e l'accettazione del sistema mentre RUP aggiunge qualcosa in più durante la prima parte di un progetto. Questa piccola aggiunta può garantire grandi dividendi se ci si rivolge ad una serie più completa di rischi.

Costruzione

L'obiettivo della costruzione è di completare lo sviluppo del sistema. La fase di costruzione è in un certo senso un processo di produzione, dove l'enfasi è posta sulla gestione delle risorse e sul controllo delle operazioni per ottimizzare costi, pianificazioni e qualità. In questo senso la tendenza di gestione subisce una transizione dallo sviluppo della proprietà intellettuale durante l'inizio e l'elaborazione, allo sviluppo di prodotti distribuibili durante la fase di costruzione e di transizione.

XP si concentra sulla fase di costruzione. È quella in cui viene prodotto il codice. Le fasi di XP hanno scopi di pianificazione, ma l'attenzione di XP è rivolta alla costruzione del codice.

Ogni iterazione di costruzione ha tre attività essenziali:

- gestione delle risorse e controllo del processo — Tutti devono sapere chi fa cosa e dove. È necessario assicurarsi che il carico di lavoro non vada oltre le proprie capacità e che stia procedendo in linea con la pianificazione.
- sviluppare ed eseguire test sui componenti — Vengono costruiti i componenti richiesti per soddisfare i casi d'uso, gli scenari ed altre funzionalità dell'iterazione. Vengono verificati con test di unità e di integrazione.

- valutare l'iterazione — Avvicinandosi al completamento dell'iterazione è necessario determinare se ne sono stati soddisfatti gli obiettivi. In caso contrario, è necessario assegnare nuovamente le priorità e gestire l'ambito per rispettare la data di consegna.

Differenti tipi di sistema richiedono tecniche diverse. RUP offre all'ingegnere software differenti linee guida ed assiste nella costruzione dei giusti componenti. I requisiti, nella forma di casi d'uso e di requisiti supplementari (non funzionali), sono descritti in modo abbastanza dettagliato per consentire all'ingegnere di svolgere il suo lavoro. Diverse attività in RUP forniscono una guida sulla progettazione, l'implementazione e la verifica di differenti tipi di componenti. Un ingegnere software esperto non ha bisogno di consultare queste attività in dettaglio. Uno meno esperto troverà invece grande aiuto per le procedure ottimali. Ogni membro del team può andare in profondità o meno nel processo secondo necessità. Tutti comunque puntano ad una singola base di conoscenza del processo.

In XP, le storie guidano l'implementazione. Nel testo Extreme Programming Installed, Jeffries, e altri dicono che le storie sono "promesse per la conversazione" con i programmatori.[1] Una comunicazione continua ed efficace è una buona procedura. Anche se ci sono sempre dettagli da chiarire, se le storie non sono abbastanza approfondite da poter permettere ai programmatori di svolgere la maggior parte del loro lavoro, esse non sono ancora pronte. I casi d'uso devono essere abbastanza dettagliati da consentire ai programmatori di implementarli. In molti casi, i programmatori aiutano nella scrittura dei dettagli tecnici del caso d'uso. Jeffries, e altri dicono inoltre che le conversazioni vengono documentate ed allegate alla storia. Anche RUP suggerisce ciò, tranne che nella forma di una specifica di caso d'uso, che può essere informale secondo necessità. La cattura e la gestione dell'esito delle conversazioni è un compito che va gestito.

La fase successiva di XP è la costruzione. Esistono perle di saggezza e di guida appropriata per gran parte dei team. Le più consistenti procedure di XP sono:

- verifica — I programmatori scrivono continuamente test per proseguire con il loro codice. I test riflettono le storie. XP raccomanda di scrivere prima i test, una procedura eccellente poiché costringe ad una approfondita comprensione delle storie e pone più domande se necessario. In ogni caso, che vengano scritti prima o dopo, l'importante è scriverli! Aggiungerli alla suite di test ed assicurarsi che vengano eseguiti ogni volta che il codice viene modificato.

- eseguire il refactoring — Ristrutturare il sistema continuamente, senza modificarne il comportamento, per renderlo più semplice o più flessibile. È necessario determinare se questa può essere una buona procedura per il proprio team. Ciò che è semplice per una persona può risultare complessa per un'altra. Esiste un esempio in cui due ingegneri molto in gamba passavano ogni sera su un progetto a riscrivere l'uno il codice dell'altro perché pensavano fosse troppo complesso. Come effetto collaterale, le build risultavano continuamente rotte il giorno successivo per il resto del team. Le verifiche aiutano, ma il team avrebbe fatto meglio a non farli sprofondare in queste guerre di codice.

- programmazione accoppiata — Secondo XP la programmazione accoppiata produce un codice migliore in meno tempo. È evidente in questo caso.[2] Esistono troppi fattori umani ed ambientali da considerare al momento di implementare questa procedura. I programmatori sono disposti a tentare questa strada? L'ambiente fisico consente a due programmatori di lavorare efficacemente su una singola workstation? Cosa bisogna fare con i programmatori che sono in telecomunicazione o in diverse locazioni?

- integrazione continua — Integrare e costruire spesso il sistema, possibilmente più di una volta al giorno. È un ottimo modo per assicurare l'integrità strutturale del codice e per consentire un controllo continuo della qualità attraverso verifiche d'integrazione.

- proprietà collettiva — Tutti possono modificare il codice in qualsiasi momento. XP conta sul fatto che una buona serie di test di unità possa ridurre i rischi insiti in questa procedura. I benefici nel rendere familiare a tutti ogni cosa non scalano oltre qualche punto - diecimila, ventimila, sicuramente meno di cinquantamila?

- progettazione semplice — Così come per il refactoring, la progettazione del sistema viene continuamente modificata per eliminare la complessità. Di nuovo, è necessario determinare quanto questa debba essere portata su una scala più grande prima che smetta di funzionare. Se si investe tempo durante la fase di elaborazione a progettare l'architettura, è plausibile credere che una progettazione semplice possa emergere e diventare stabile in meno tempo.

- standard di codifica — Questa è sempre una buona procedura. Non importa quali siano gli standard finché ci sono e tutti sono concordi nell'utilizzarli.

RUP e XP concordano sul fatto che le iterazioni debbano essere gestire (o controllate). Le metriche possono fornire buone informazioni di pianificazione poiché
aiutano nella scelta di quella migliore per il proprio team. Ci sono tre cose da misurare: il tempo, la dimensione ed i difetti. Da ciò è possibile

[1] Questa descrizione è attribuita a Allistair Cockburn.

[2] Strengthening the Case for Pair Programming, IEEE Software, Luglio/Agosto, 2000.

ottenere ogni sorta di statistica interessante. XP fornisce semplici metriche da usare per determinare il progresso e prevedere il conseguimento. Queste metriche sono incentrate sulla quantità di storie finite, di test passati e dei trend in queste statistiche. XP insiste nell'utilizzare una piccola quantità di metriche in quanto cercare di utilizzarne di più non migliora necessariamente le possibilità di successo per il progetto. RUP fornisce linee guida su cosa misurare e come nonché esempi di metriche. In ogni caso, le metriche devono essere semplici, obiettive, facili da raccogliere e da interpretare e difficili da fraintendere.

Quali artefatti appaiono durante la fase di costruzione? In base alle iterazioni, le prime o le ultime della fase di costruzione, è possibile creare quanto segue:

- Un componente — Un componente rappresenta una parte di codice software (fonte, binaria o eseguibile) o un file contenente informazioni; ad esempio, un file di avvio o di ReadMe. Un componente può anche essere un aggregato di altri componenti, come un'applicazione costituita da diversi elementi eseguibili.
- Materiali di formazione — In base ai casi d'uso, produrre una bozza di manuali utente e altri materiali di formazione il prima possibile se il sistema possiede un forte aspetto d'interfaccia utente.
- Un piano di distribuzione — Il cliente ha bisogno di un sistema. Il piano di distribuzione descrive i compiti necessari per installare, verificare ed eseguire efficacemente la transizione del prodotto alla comunità degli utenti. Per i sistemi incentrati sul web il piano di distribuzione si è rivelato sempre più importante.
- Un piano d'iterazione della fase di transizione — Quando si avvicina il momento di distribuire il software agli utenti, viene completato ed analizzato il piano d'iterazione della fase di transizione.

È davvero tutto in base al codice?

Esistono altre differenze rispetto a quelle relative all'approccio all'architettura tra RUP e XP. Una riguarda il modo di comunicare la progettazione. XP indica che il codice è la progettazione e viceversa. Vero è che il codice è sempre in accordo con sé stesso. Investire impegno nella cattura e nella comunicazione della progettazione diversa dal codice, viene considerato tempo ben speso. Questa breve storia può evidenziare quanto appena detto.

Un ingegnere aveva due passate esperienze su progetti software in cui la progettazione risiedeva nel codice, l'unico posto in cui trovare informazioni su di essa. Entrambi i progetti erano relativi ad un compilatore: uno per migliorare e mantenere un programma di ottimizzazione per un compilatore Ada, l'altro per portare il front-end di un compilatore su una nuova piattaforma e collegare ad essa un generatore di codice sviluppato da terze parti.

La tecnologia di un compilatore è complessa, ma ben nota. Su entrambi i progetti, l'ingegnere ha voluto una panoramica di progettazione ed implementazione del compilatore (o programma di ottimizzazione). In ogni caso ha ricevuto una pila di elenchi di codici sorgente, diversi pollici compatti e gli fu detto di “cercare lì dentro.” Avrebbe dato tutto per pochi diagrammi ben costruiti con un pò di testo a supporto. Il progetto del programma di ottimizzazione non è stato mai completato. Il progetto del compilatore ha avuto un esito positivo, con un codice di grande qualità a causa di serie intensive di test sviluppati insieme ad esso. L'ingegnere ha speso giorni interi a navigare per il codice in un debugger per cercare di capire cosa avesse fatto. I costi personali di un minore burn-out e quelli sul team non sono stati convenienti. Non c'è stata la possibilità di fermarsi dopo 40 ore, come XP suggerisce ed è stato impiegato uno sforzo enorme per completare il lavoro.

Il problema principale quando si ha solo un codice - non importa quanto ben documentato - è che non spiega quale problema risolve ma si limita solo a comunicarne la soluzione. Alcune documentazioni dei requisiti impiegano molto per spiegare gli obiettivi originari e molto dopo che i primi utenti e sviluppatori sono andati via. Per mantenere un sistema, è spesso necessario capire cosa avesse in mente il team originario. Alcuni documenti di progettazione di alto livello sono simili - spesso il codice si trova ad un livello troppo basso di astrazione per poter dire realmente cosa un sistema, nel suo complesso, stia cercando di fare. Questo è vero in particolare nei sistemi Object-Oriented in cui il thread di esecuzione è difficile se non impossibile da seguire solo mediante l'osservazione delle classi. I documenti di progettazione indicano dove guardare nel caso di problemi successivi - che ci sono sempre.

Il morale della storia è che investire del tempo nella cattura e nel mantenimento dei documenti di progettazione è davvero utile. Riduce il rischio di fraintendimento e può velocizzare lo sviluppo. L'approccio di XP consiste nell'impiegare qualche minuto nell'abbozzare la progettazione o nell'usare le carte di CRC.[1]Il team non le mantiene, ma inizia a lavorare sul codice. Vi è l'assunzione che i compiti siano semplici abbastanza al punto che già si sa come procedere. Ma anche se questo è vero, chi arriva in un secondo momento potrebbe non essere così fortunato. RUP suggerisce di spendere più tempo nella cattura e nel mantenimento di questi artefatti di progettazione.

[1] Le carte CRC (Classe, Responsibilità e Collaborazione), sviluppate da Kent Beck e Ward Cunningham, insegnano ai professionisti i principi della progettazione orientata all'oggetto.

Transizione

Il punto focale della fase di transizione consiste nell'assicurare che il software sia disponibile per i suoi utenti finali. La fase di transizione include la verifica del prodotto in preparazione per il rilascio e l'esecuzione di rettifiche di minor importanza basate sul feedback dell'utente. A questo punto nel ciclo di vita, il feedback dell'utente dovrebbe essere maggiormente concentrato sulla rifinitura del prodotto e sulle problematiche di configurazione, installazione e usabilità.

Rilasciare presto e spesso è un'ottima idea. Ma cosa si intende per rilascio? XP è vago al riguardo e non indica le problematiche di fabbricazione delle problematiche necessarie per il rilascio di software commerciale. Può essere possibile abbreviare alcune delle problematiche in un progetto interno; ma anche in questo caso è necessaria documentazione, formazione e così via. Cosa succede nel caso di supporto e gestione delle modifiche? È realistico aspettarsi che il cliente possa controllare anche queste sul posto? Bruce Conrad fa notare nella sua revisione InfoWorld di XP[1] che i clienti possono non gradire un software in continua evoluzione. Bisogna misurare rapidamente i benefici delle modifiche per il cliente rispetto allo svantaggio ed alla possibile destabilizzazione che ne può derivare.

Quando viene deciso il rilascio, è necessario fornire all'utente finale più che un semplice codice. Le attività e gli artefatti nella fase di transizione fungono da guida verso questa parte del processo di sviluppo software. Le attività si concentrano sul conseguimento di un prodotto utilizzabile per il cliente. Le attività essenziali della fase di transizione sono:

- finalizzare il materiale di supporto per l'utente finale — Questa attività può essere semplice come contrassegnare gli elementi, ma è necessario assicurarsi che l'organizzazione sia preparata per il supporto al cliente.
- verifica del prodotto distribuibile in un ambiente del cliente — Se si ha la possibilità di simulare l'ambiente del cliente nel proprio, è consigliabile farlo. Altrimenti, andare dal cliente ed installare il software per verificarne il funzionamento. Non è consigliabile trovarsi nella poco invidiabile posizione di dover dire al cliente che “funzionava sul nostro sistema”.
- regolare il prodotto sulla base del feedback del cliente — Se possibile, pianificare uno o più periodi di verifica beta in cui consegnare il software ad un limitato numero di clienti. Nel caso, è necessario gestire il periodo di verifica beta e considerare il feedback del cliente alla “fine dei giochi”.
- consegnare il prodotto finale all'utente finale — In base al tipo di software e di rilascio, esistono molti dettagli sull'imballaggio, la fabbricazione ed altre questioni produttive da affrontare. Di rado è possibile inserire il software in una directory ed inviare una mail facendo così sapere alla comunità dei clienti che il software è pronto.

Come per la maggior parte delle altre fasi, il rigore e la complessità del processo può variare. Tuttavia, se non si pone attenzione ai dettagli della distribuzione, è possibile annullare settimane e mesi di impegno per un buon sviluppo e finire con un prodotto fallace nel mercato di riferimento.

È possibile produrre diversi artefatti durante la fase di transizione. Se il prodotto è destinato a futuri rilasci (e come non potrebbe?), bisogna iniziare ad identificare le caratteristiche ed a correggere i difetti per il prossimo rilascio. Gli artefatti essenziali di ogni progetto sono:

- un piano di distribuzione — Finalizzare il piano di distribuzione avviato nella fase di costruzione ed utilizzarlo come percorso per la consegna al cliente.
- note di rilascio — È un raro prodotto software che non ha istruzioni dell'ultimo minuto per l'utente finale. Pianificare su di esso ed ottenere un formato fruibile e coerente per le proprie note.
- documentazione e materiali di formazione — Questi materiali possono assumere un ampio spettro di forme. Viene fornito tutto on-line? Esiste una formazione? La guida al prodotto è completa ed utilizzabile? Non si pensi che il cliente sappia ciò che sanno gli sviluppatori. Il successo dipende dall'aiutarli ad avere successo.

Riepilogo

Costruire software è molto più che scrivere un codice. Un processo di sviluppo software deve focalizzarsi su tutte le attività necessarie per garantire qualità al cliente. Un processo completo non deve essere pesante. È stato mostrato come sia possibile ottenere un progetto piccolo ma completo concentrando l'attenzione sulle attività e gli artefatti del progetto. Eseguire un'attività o produrre un artefatto se può aiutare a diminuire i rischi sul progetto. Utilizzare poco o molto processo e rigore secondo le necessità del team di progetto e dell'organizzazione.

[1] http://www.infoworld.com/articles/mt/xml/00/07/24/000724mtextreme.xml

RUP e XP non sono necessariamente esclusivi. Mettendo insieme tecniche da entrambi i metodi, è possibile giungere ad un processo che favorisce la consegna di software di qualità migliore più rapidamente di quanto non si faccia oggi. Robert Martin descrive un processo chiamato dX , che dichiara essere rispettoso di RUP.[1] È un'istanza di un processo costruito dal framework di RUP.

Un buon processo software incorpora le procedure ottimali verificate dall'industria. Le procedure ottimali sono quelle testate nel tempo, in reali organizzazioni di sviluppo software. XP è un metodo su cui viene focalizzata oggi l'attenzione. È incentrato sul codice ed offre una promessa di un minimo carico di processo e di massima produttività. Esistono molte tecniche in XP che consentono la considerazione e l'adozione nelle giuste situazioni.

XP si concentra sulle storie, i test ed il codice - discute sulla pianificazione da una certa distanza, ma tratta in modo chiaro la cattura dei piani. XP implica che è possibile produrre altro da “fai una progettazione CRC con poche carte o fai una bozza di qualche UML…” oppure “Non produrre documenti o altri artefatti che non vengono utilizzati…”, ma li tratta nel passaggio. RUP invita a considerare solo la produzione di ciò che è utile e richiesto quando viene formulato ed aggiornato il piano di sviluppo ed identifica quali possono essere questi elementi.

RUP è un processo che si indirizza al completo ciclo di vita di sviluppo software. Si focalizza sulle procedure ottimali evolute da migliaia di progetti. Viene incoraggiata l'indagine e l'invenzione di nuove tecniche che conducono alle procedure ottimali. Nel momento in cui emergono migliori nuove procedure, si cerca di incorporarle in RUP.

Appendice A: Rational Unified Process

Rational Unified Process o RUP fornisce un approccio disciplinato allo sviluppo software. È un prodotto del processo , sviluppato e mantenuto da Rational Software. Porta con sé diversi percorsi pronti all'uso per differenti tipi di progetti software. RUP fornisce inoltre informazioni per aiutare l'utente ad utilizzare i tool di Rational per lo sviluppo software, ma non li richiede per un applicazione efficace di un'organizzazione; l'integrazione con altre offerte dei venditori è possibile.

RUP fornisce una guida per tutti gli aspetti di un progetto software. Non richiede l'esecuzione di particolari attività o la produzione di particolari artefatti. Fornisce informazioni e linee guida per decidere cosa è applicabile in un'organizzazione. Fornisce inoltre linee guida che aiutano a personalizzare il processo se nessuno dei percorsi pronti all'uso si adatta ad uno specifico progetto o organizzazione.

RUP enfatizza l'adozione di determinate buone procedure del moderno sviluppo software, come modo per ridurre i rischi inerenti allo sviluppo di nuovo software. Queste procedure ottimali sono:

1. Sviluppare in modo iterativo
2. Gestire requisiti
3. Utilizzare strutture basate su componenti
4. Modellare visivamente
5. Verificare continuamente la qualità
6. Controllare modifiche

Queste procedure ottimali sono intrecciate con le definizioni Rational Unified Process

di:

- Ruoli — serie di attività eseguite ed artefatti posseduti.
- Discipline — aree di attenzione dell'impegno della programmazione software come i requisiti, l'analisi e la progettazione, l'implementazione e il test.
- Attività — definizioni del modo in cui vengono prodotti e valutati gli artefatti.
- Artefatti — i prodotti del lavoro usati, prodotti o modificati nelle prestazioni delle attività

RUP è un processo iterativo che identifica quattro fasi per ogni progetto di sviluppo software. Nel tempo, il progetto passa attraverso le fasi di inizio, elaborazione, costruzione e transizione. Ogni fase contiene una o più iterazioni in cui

[1] http://www.objectmentor.com/publications/RUPvsXP.pdf. Questo è un capitolo del testo non pubblicato da Martin, Booch e Newkirk.

viene prodotto un sistema eseguibile anche se probabilmente incompleto (tranne forse nella fase di inizio). Durante ogni iterazione vengono eseguite attività da diverse discipline a livelli variabili di dettaglio. Il seguente è un diagramma di panoramica di RUP.

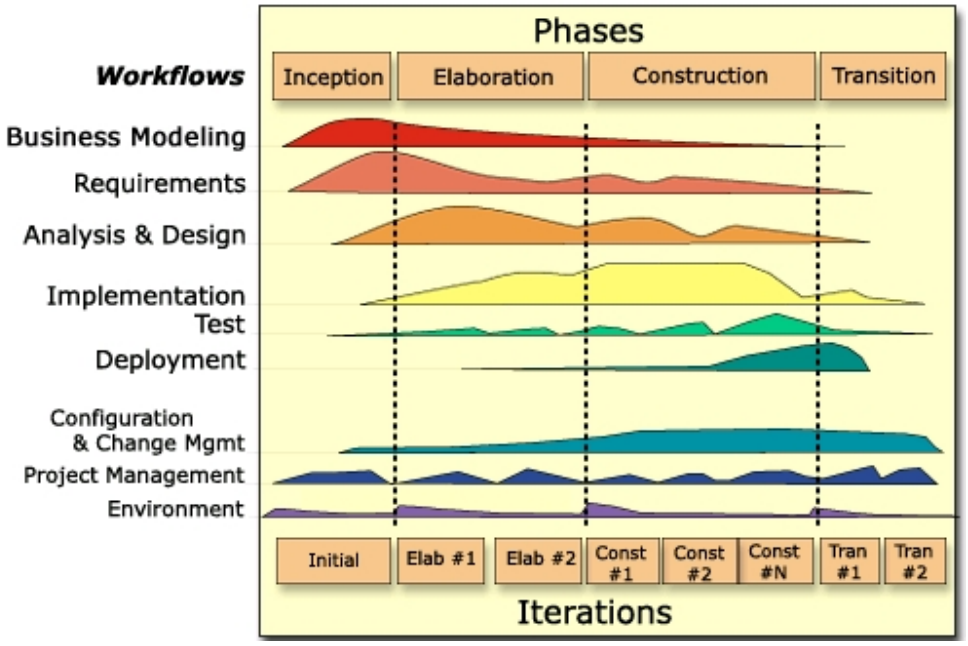


Diagramma di panoramica di RUP.

The Rational Unified Process, An Introduction, Second Edition è una buona descrizione di RUP. È possibile trovare ulteriori informazioni nonché una valutazione di RUP sul sito web di Rational Software web site: www.rational.com.

Appendice B: eXtreme Programming

eXtreme Programming (XP) è una disciplina di sviluppo software sviluppata da Kent Beck nel 1996. Si basa su quattro valori: comunicazione, semplicità, feedback e coraggio. Accentua una comunicazione continua tra il cliente e i membri del team di sviluppo in quanto il primo è in sede mentre lo sviluppo va avanti. Il cliente in sede decide cosa deve essere costruito ed in che ordine. È necessario garantire sempre la semplicità eseguendo continuamente il refactoring del codice e producendo un insieme minimo di artefatti non di codice. I meccanismi feedback consistono in molti brevi rilasci e continue verifiche di unità. Per coraggio si intende il fare la cosa giusta, anche se non molto popolare. Significa essere onesti su cosa è possibile fare e cosa no.

Dodici procedure di XP supportano i quattro valori. Queste sono:

- Il planning game — determinare le caratteristiche nel rilascio successivo attraverso una combinazione di storie elencate per importanza e valutazioni tecniche.
- Piccoli rilasci — Rilasciare spesso il software al cliente con piccole versioni incrementali.
- Metafora — È una semplice storia condivisa o descrizione su come funziona il sistema.
- Progettazione semplice — Mantenere semplice la progettazione mantenendo semplice il codice. Cercare continuamente la complessità nel codice e rimuoverla in una volta sola.
- Varifica — Il cliente scrive test per verificare le storie. I programmatori scrivono test per verificare qualunque cosa possa danneggiarsi nel codice. I test vengono scritti prima del codice.
- Refactoring — È una tecnica di semplificazione per rimuovere duplicazioni e complessità dal codice.

 Programmazione accoppiata — Team composti da due programmatori con un singolo computer sviluppano l'intero codice. Uno lo scrive oppure lo dirige, mentre l'altro lo analizza a fini di correttezza e comprensibilità insieme.

 Proprietà collettiva — Ognuno detiene tutto il codice. Questo significa che ognuno ha la capacità di modificarlo in ogni momento.

 Integrazione continua — Costruire ed integrare il sistema diverse volte al giorno quando il compito d'implementazione è completato.

 Forty-hour week — I programmatori non possono lavorare alla massima efficienza se sono stanchi. Lo straordinario non è mai consentito per due settimane consecutive.

 Cliente in sede — Un vero cliente lavora nell'ambiente di sviluppo a tempo pieno per dare una mano a definire il sistema, scrivere test e rispondere alle domande.

 Standard di codifica — I programmatori adottano un coerente standard di codifica.

Esistono tre libri attualmente disponibili su XP:

1. eXtreme Programming Explained
2. Extreme Programming Installed
3. Planning Extreme Programming

Sono disponibili diversi siti web per ulteriori informazioni su XP.

Rational®
the software development company

Sedi:

Rational Software
18880 Homestead Road
Cupertino, CA 95014
Tel: (408) 863-9900

Rational Software
20 Maguire Road
Lexington, MA 02421
Tel: (781) 676-2400

numero verde: (800) 728-1212
E-mail: info@rational.com
web: www.rational.com
Ubicazione internazionale: www.rational.com/worldwide

Rational, il logo Rational e Rational Unified Process sono marchi registrati di Rational Software Corporation negli Stati Uniti e/o altri paesi. Microsoft, Microsoft Windows, Microsoft Visual Studio, Microsoft Word, Microsoft Project, Visual C++, e Visual Basic sono marchi o marchi registrati di Microsoft Corporation. Tutti gli altri nomi vengono utilizzati solo per fini di identificazione e sono marchi delle rispettive società. TUTTI I DIRITTI RISERVATI. Realizzato in U.S.A.

Copyright 2002 Rational Software Corporation.
Soggetto a modifiche senza notazione.