Ciclo e Processo di Sviluppo: approcci tradizionali, evolutivi, agili, free open source software

Ingegneria del software

"L'istituzione e l'impiego di principi ingegneristici fondati, allo scopo di ottenere in modo economico software affidabile ed efficiente su macchine vere"

F. Bauer, Conferenza NATO Garmish 1968

" (1) Applicazione di una strategia sistematica, disciplinata e misurabile allo sviluppo, esercizio e manutenzione del software; cioè applicazione dell'ingegneria del software.
(2) Studio delle strategie di cui al punto (1)."

Definizione IEEE (Institute of Electrical and Electronics Engineers)

Il software è:

- **intangibile**
- **non definibile a priori**
- **spesso costruito ad hoc (più che assemblato)**
- **ad alta intensità di lavoro umano**
- **soggetto al cambiamento**

Ingegneria del software

- Qual è il problema da risolvere?
- Quali le caratteristiche dell'oggetto che risolve il problema
- Come lo realizzo?
- Come ne garantisco la rispondenza al problema da risolvere?
- Come evolve nel tempo?

"L'ingegneria del software si occupa dello studio dei processi relativi alla concezione, sviluppo, deployment, evoluzione ed utilizzo di prodotti software. E' una disciplina complessa dove confluiscono aspetti tecnologici, organizzativi, cognitivi. Proprio per questo motivo la qualità delle risorse umane gioca un ruolo fondamentale"

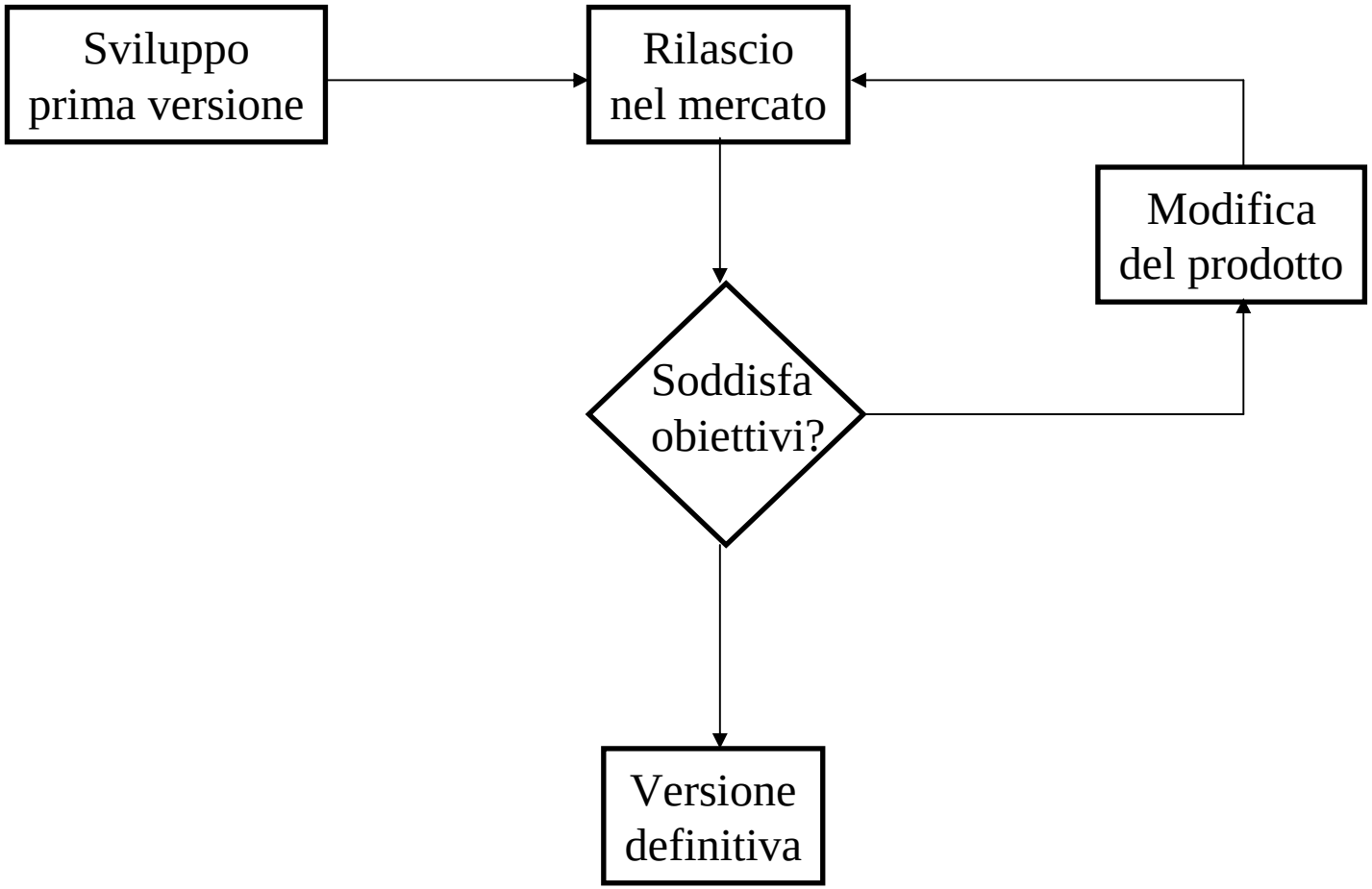
CEFRIEL, WebBook dell'Ingegneria del Software

Modello di sviluppo

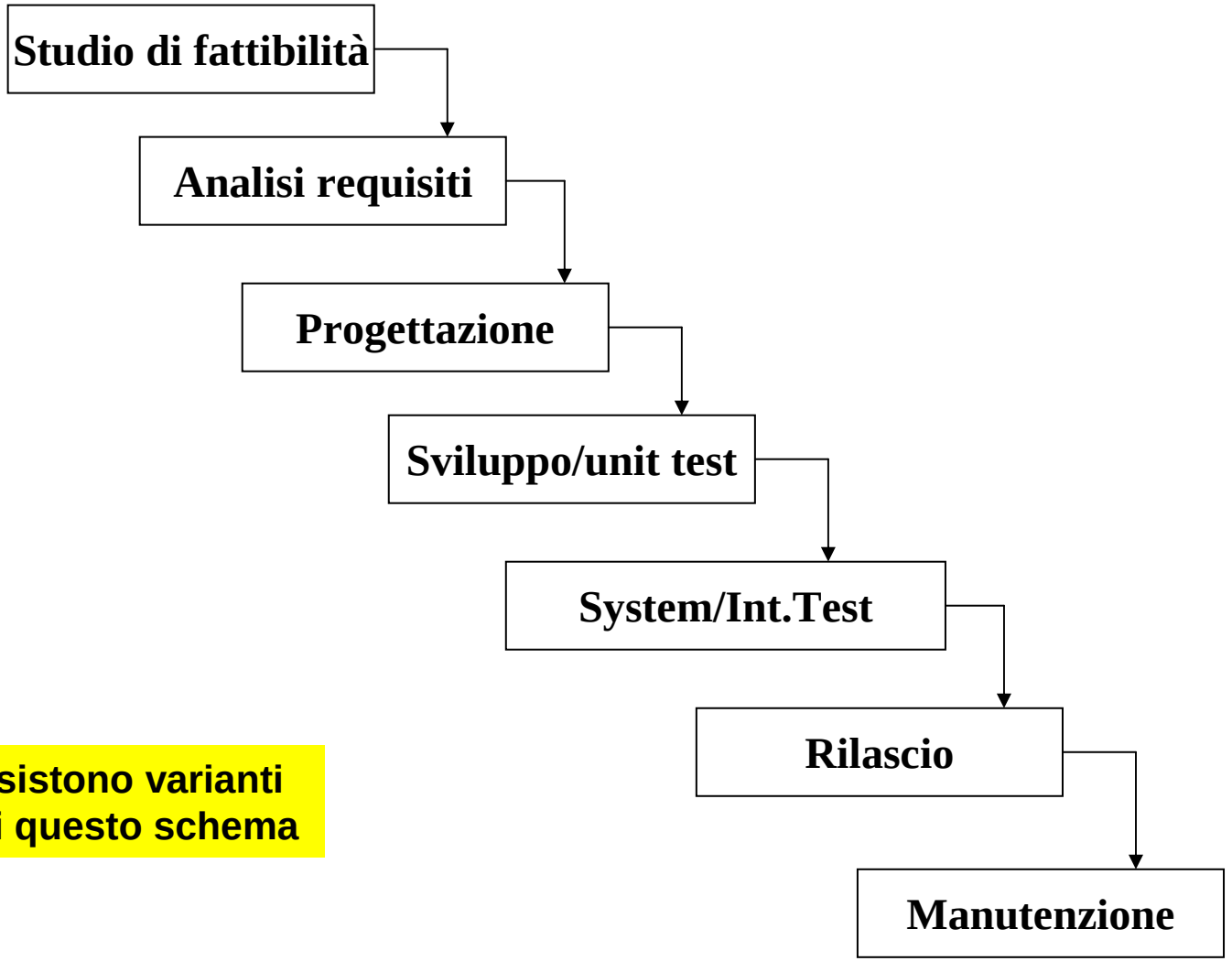
6 buoni motivi per:

- affrontare lo sviluppo software in presenza di:
  - requisiti “non certi”
  - ambiente di sviluppo “non certo”
  - ambiente di erogazione “certo e non adattabile”
  - risorse umane creative ed indipendenti
- ridurre i costi di sviluppo (utilizzo ripetuto di tools e standard)
- rispettare tempi e costi di sviluppo
- favorire il riuso (componenti, conoscenza)
- migliorare il processo con l’adozione di un approccio ripetibile
- crescere assieme: senza metodo, emerge solo chi possiede particolari “caratteristiche naturali” (metafora dello sport)

Build and Fix



Waterfall



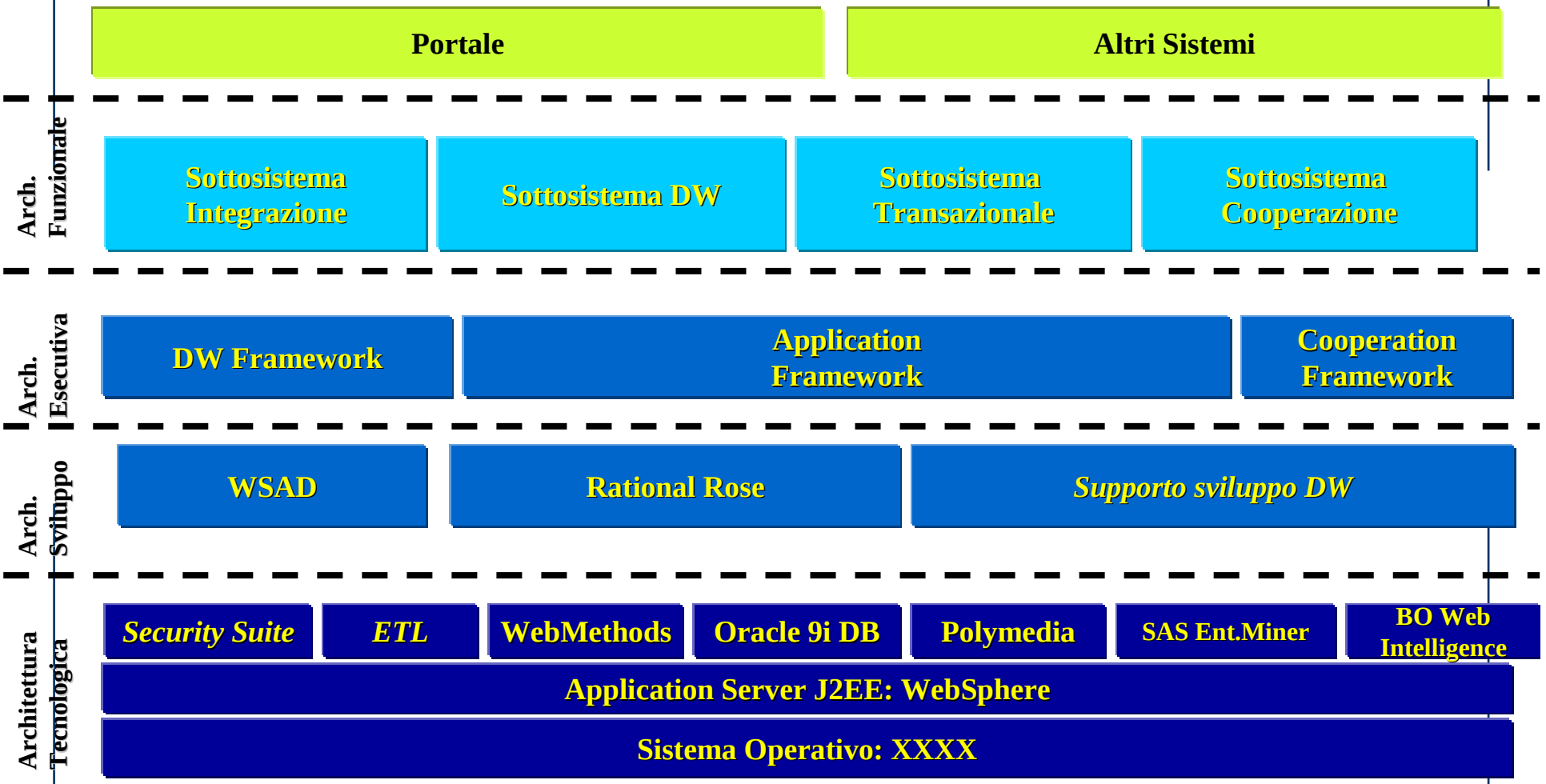
Esistono varianti di questo schema

Analisi dei requisiti

- **Identificazione dei requisiti**
  - **impegni ed esigenze che richiedono un'implementazione**
- **Classificazione dei requisiti**
  - **suddivisione in categorie per facilitare individuazione e tracciatura**
- **Valutazione dei requisiti**
  - **modalità di verifica e validazione**
- **Tracciabilità dei requisiti**
  - **dove e come il requisito trova attuazione (registrazione di tutte le attività/deliverables che lo implementano)**

Disegno Architetturale

Individua e dettaglia “le architetture” del sistema



Codifica

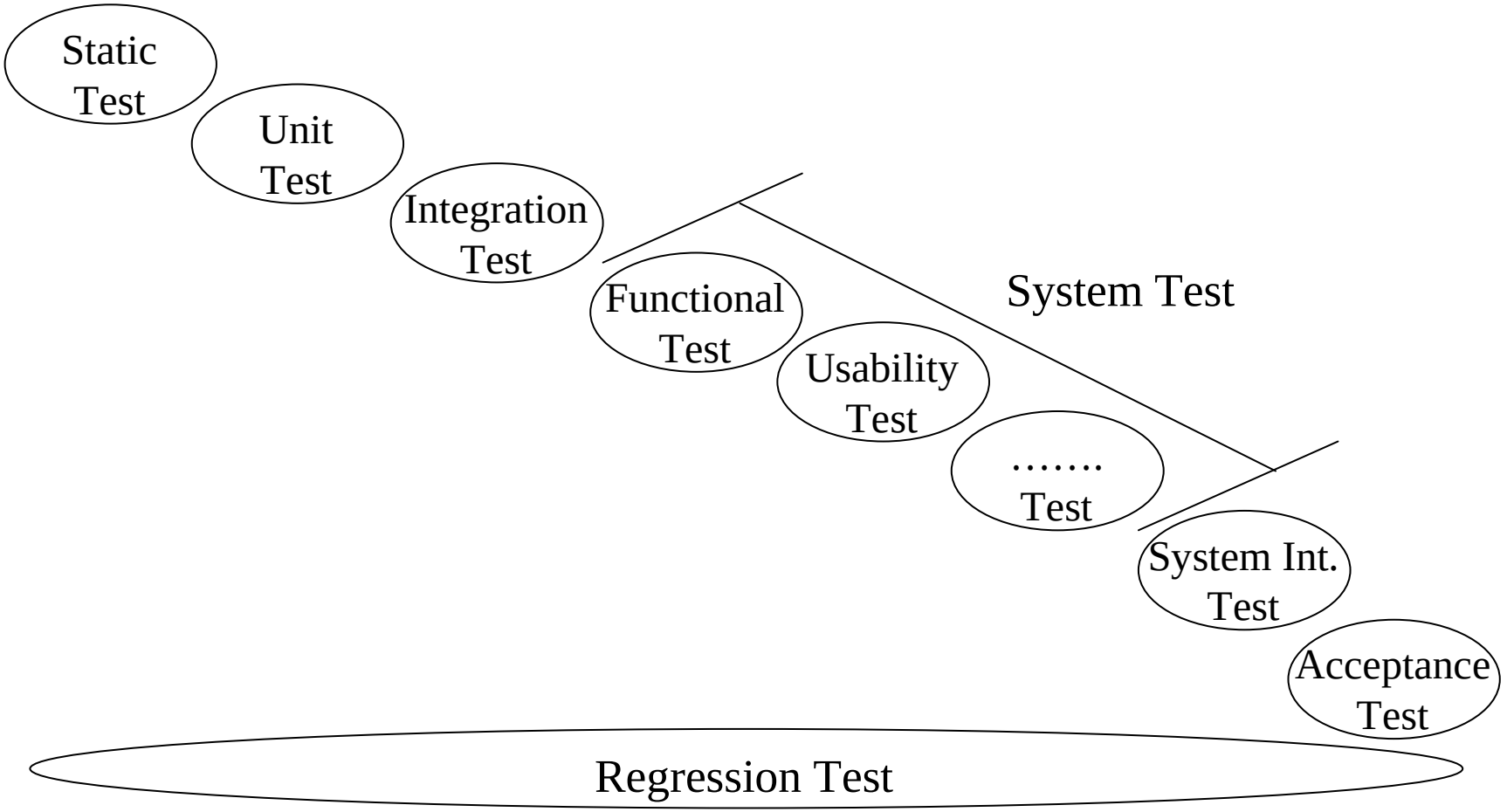
Attività da realizzare:

- Interfaccia grafica (usabilità, accessibilità)
- Codifica
- Commenti al codice (javadoc)
- Test singoli componenti
- Bozza di manuale utente
- Verifica di integrità in sede di VCS o build

Best practices:

- Test-first-development (Junit)
- Riuso
- Refactoring
- Code review

Test



Collaudo

Attività da realizzare:

- Installazione in ambiente di collaudo
- Test di accettazione o collaudo

Sotto-insieme del system test:

con diversa attenzione a

- analisi del contesto (individuazione fattori determinanti)
- determinazione del livello di copertura
- verifica del livello di copertura

Punto di vista del Cliente

Waterfall

- **Waterfall, linear sequential model**
- **Definito ed adottato all'inizio degli anni '70 (reazione al *code and fix*)**
- **E' un riferimento per tutti gli altri processi di sviluppo**
- **Suggerisce un approccio sistematico e sequenziale allo sviluppo**
  - ✓ **la fase successiva non inizia se non è terminata la precedente**
  - ✓ **ogni fase produce un risultato finito**
  - ✓ **ogni fase è soggetta ad attività di controllo**
  - ✓ **assenza di ricicli e nullo (o minimo) overlap tra le fasi**
- **I requisiti devono essere ben definiti e stabili**
- **L'utente vede il risultato solo alla fine del processo**
- **Varianti del modello dipendono dal rapporto Cliente/Fornitore**

Waterfall

E’ stato storicamente adottato per:

- ✓ **diminuire i costi di sviluppo (rispetto al *build & fix*)**
- ✓ **la convinzione che fosse possibile progettare correttamente l’applicazione da subito**
- ✓ **sostanziale stabilità dei requisiti per le applicazioni dell’epoca**
- ✓ **mancato riconoscimento dell’evoluzione del software**

Waterfall

Punti di forza

- ❑ Ha definito molti concetti utili
- ❑ Facilmente comprensibile ed applicabile
- ❑ Riflette la struttura dei contratti
- ❑ Ogni fase è sottoposta a controllo
- ❑ Permette di individuare chiare milestones nel progetto

Punti di debolezza

- ❑ Rende difficile gestire l'instabilità dei requisiti
- ❑ L'interazione con il cliente è solo all'inizio ed alla fine
- ❑ I ricicli esistono
- ❑ Ritarda l'individuazione dei problemi
- ❑ I deliverables intermedi non rappresentano il prodotto finale
- ❑ Un ritardo in ogni fase si ripercuote sull'intero processo

Waterfall

Quando utilizzarlo

- In presenza di requisiti stabili
- Quando si hanno buone competenze sul dominio
- In un contesto tecnologico maturo

Quando NON utilizzarlo

- Software complesso e contesto innovativo
- Progetti di lunga durata
- Requisiti non ben definiti all'inizio
- Tecnologia innovativa

Approcci evolutivi

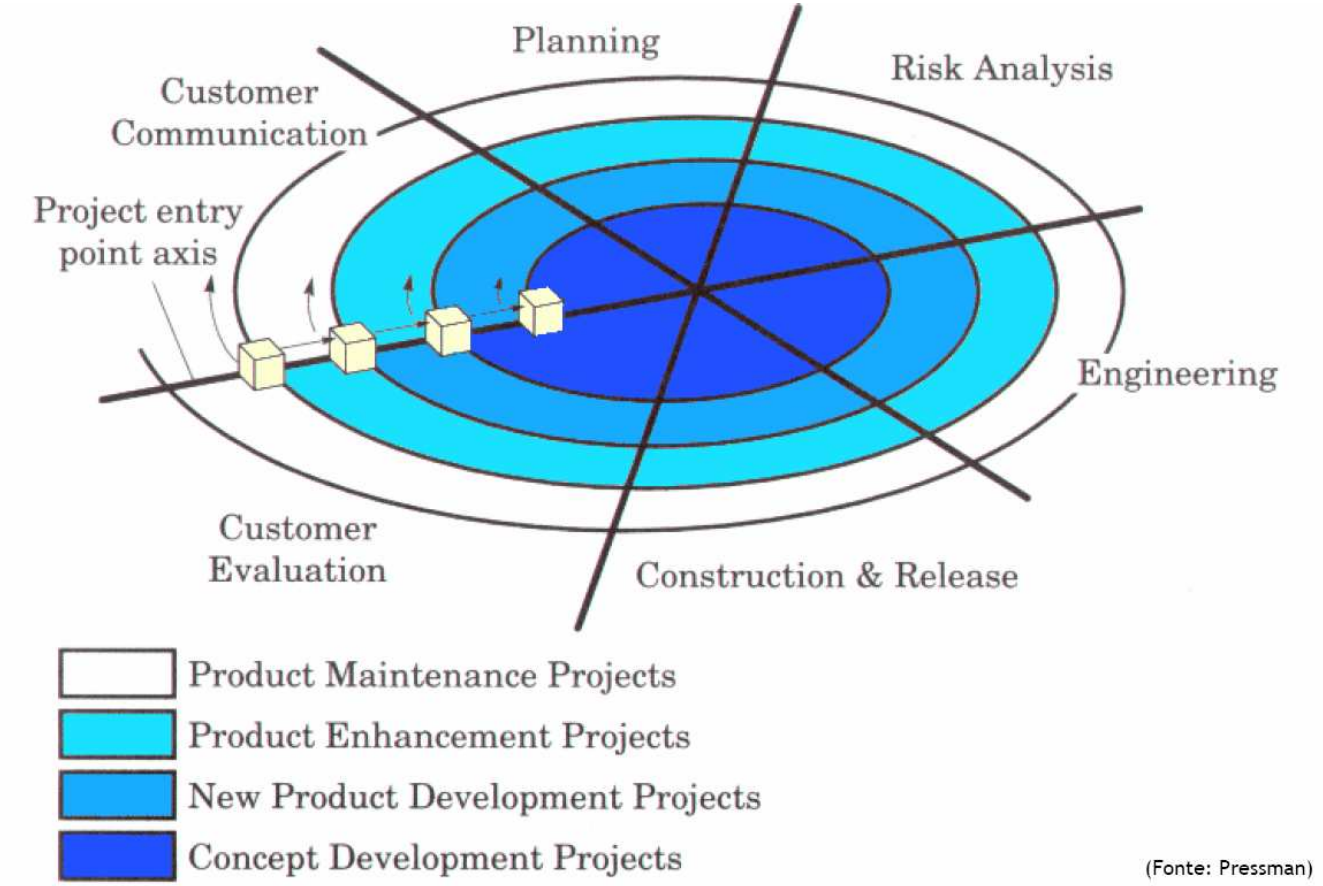
- **Riconoscono:**
  - ✓ L'inevitabilità delle evoluzioni dei requisiti, delle tecnologie, del mercato, …
  - ✓ L'impossibilità di arrivare a realizzare l'intera applicazione in un unico passo
- **Prevedono il rilascio di versioni successive, con funzionalità parziali, per far fronte a:**
  - ✓ Necessità di seguire le inevitabili evoluzioni
  - ✓ Pressioni del mercato e dei concorrenti

- **Vantaggio: un "primo prodotto finito" è disponibile per l'utente in tempi "rapidi"**
- **Svantaggio: Il "prodotto finale" può richiedere un maggiore tempo ed effort di sviluppo rispetto all'approccio "a cascata" (affermazione vera in caso di requisiti stabili)**

Modello a spirale (Bohem)

Riconcilia le caratteristiche fondamentali del modello evolutivo

- Sviluppo in una serie di incrementi
- I primi incrementi anche “su carta” (es. 1°specifica iniziale)



(Fonte: Pressman)

Modello a spirale (Bohem)

Ogni incremento è una ripetizione ciclica di task region:

- **Customer communication (Comunicazioni con il cliente):**
  - ✓ Attività volte a stabilire un efficace colloquio tra il cliente e il team di sviluppo
- **Planning (Pianificazione):**
  - ✓ Raccolta requisiti e definizione piano di progetto (risorse e scadenze)
- **Risk analysis (Analisi dei rischi):**
  - ✓ Stima e prevenzione dei rischi tecnici e di gestione
- **Engineering (Strutturazione)**
  - ✓ Costruzione di rappresentazioni dell'applicazione da sviluppare
- **Construction & release (Costruzione e rilascio)**
  - ✓ Realizzazione, collaudo e installazione
- **Costumer evaluation (Valutazione da parte del cleinte):**
  - ✓ Rilevazione delle reazioni da parte del cliente

Modello a spirale

- **Ogni task region è ulteriormente scomposta in *task***
  - ✓ Numero di task e formalità nella loro definizione dipendenti dal tipo di progetto
- **A ogni iterazione si ripetono i task relativi a ciascuna task region**
  - ✓ es. numero di iterazioni, pianificazione, schedule, stime dei task successivi sono ricalcolati ad ogni passaggio dalla task region *planning*
- **Attività trasversali: *configuration management*, *quality assurance*, ...**
- **I cicli della spirale possono corrispondere anche alle evoluzioni "post-rilascio"**
- **Adotta un approccio *risk-driven* al processo di sviluppo del software**
- **Fa uso della prototipazione per ridurre i rischi (in qualsiasi momento, non solo nelle prime iterazioni)**
- **Ciascuna iterazione adotta l'approccio sistematico del ciclo di vita a cascata**

Modello a spirale

Punti di forza

- Utilizza i vantaggi della prototipazione ad ogni ciclo
- Si presta alla gestione di progetti innovativi o complessi
- Focalizza l'attenzione sugli obiettivi di business
- I rischi sono presi in considerazione ad ogni stadio del processo

Punti di debolezza

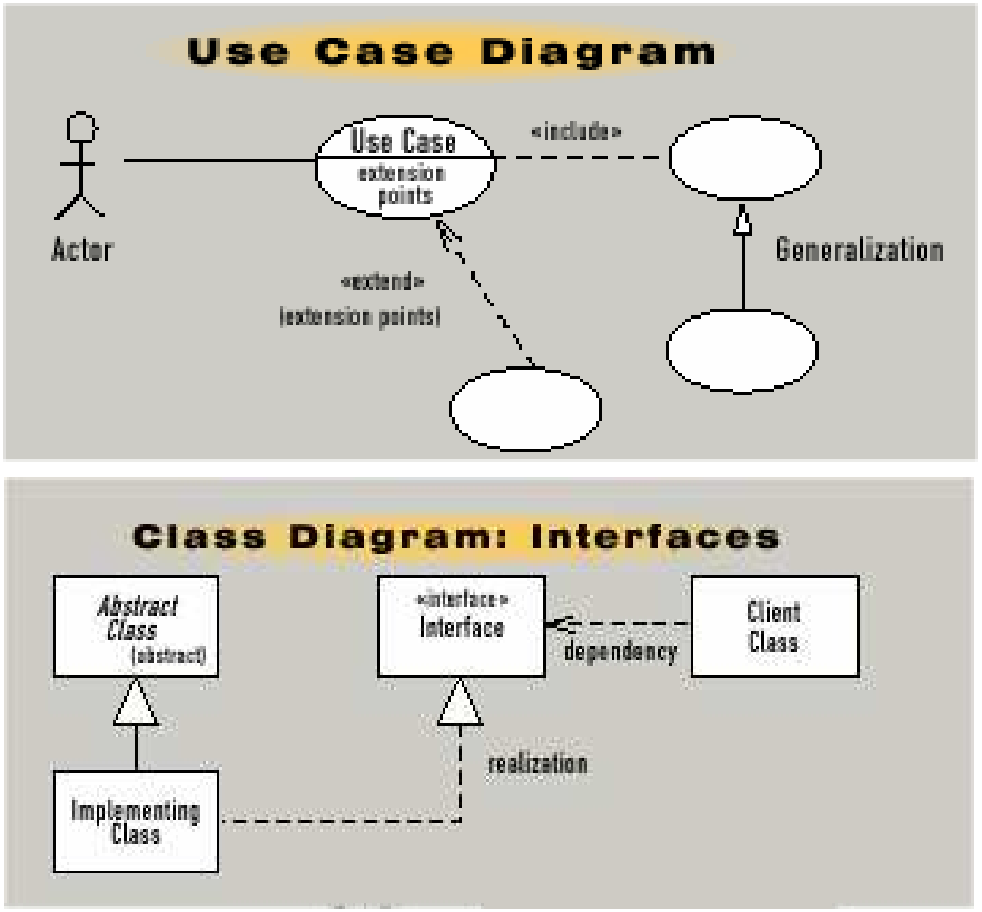
- Il modello esige competenze ad alto livello per la gestione dei rischi
- Il controllo della strategia evolutiva può diventare difficoltoso
- Mancanza di dati statistici sulla sua applicazione

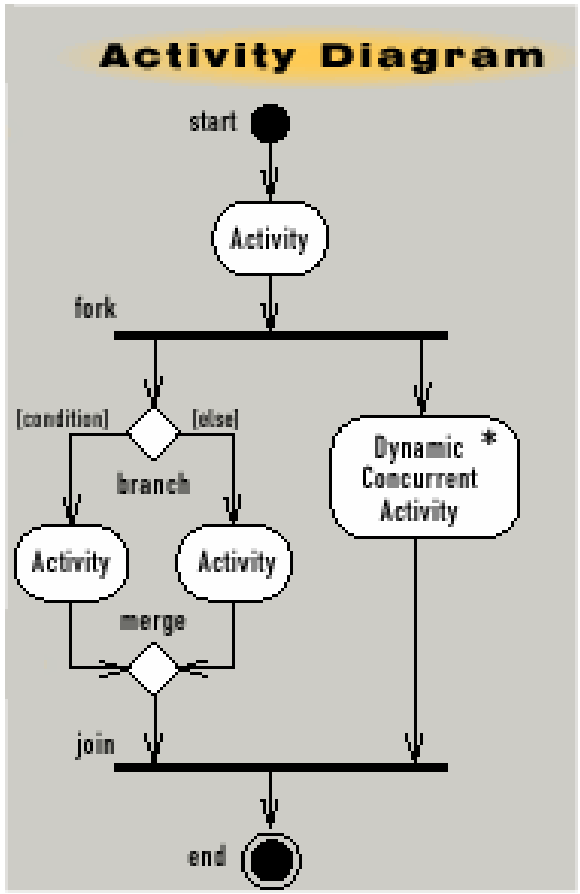
Unified Process

- **E' un processo "configurabile": definisce un "framework" o "generic process" che può essere specializzato per diversi contesti**
- **E' guidato dagli *use case* e dal rischio**
- **Utilizza UML come LINGUAGGIO per la specifica di molti elaborati (artefatti) del processo**
- **Incentrato sull'architettura (l'approccio è quello di realizzare ed implementare fino dalle *fasi iniziali del progetto un robusto studio architetturale)***
- **Basato su "Best Practices": approcci allo sviluppo di sistemi software largamente utilizzati con successo nell'industria**
- **Dispone di linee guida, template e guide all'uso**
- **E' supportato da tool CASE che consentono di automatizzare la creazione dei modelli utilizzati nel processo e la l'attività di produzione della documentazione**

Unified Process: UML

Esempi di artefatti in UML (Unified Modeling Language) fonte: www.uml.org



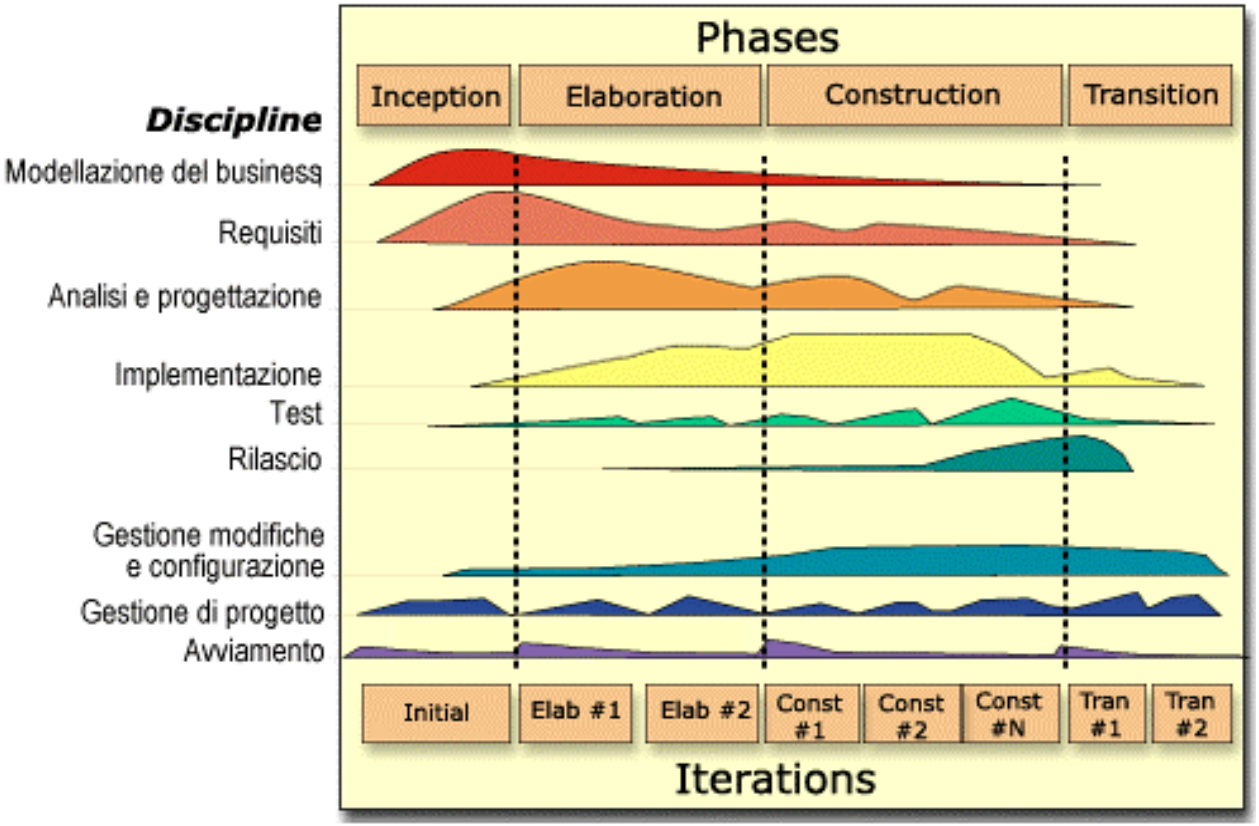


Unified Process

- **Il ciclo di vita di riferimento è la spirale di Boehm**
  - **Iterativo:** scompone il progetto in una serie di progetti più piccoli, o mini-progetti, ciascuno dei quali realizza un sistema funzionante ed autocontenuto, anche se incompleto
  - **Incrementale:** sistema sempre più completo, fino ad arrivare, con l'ultima iterazione, alla realizzazione del sistema completo
  - **Adattivo:** il suo svolgersi deve sempre essere "adattato" al contesto in cui si inserisce, allo scopo di raggiungere l'obiettivo per cui è utilizzato: la "qualità" del prodotto
- **E' strutturato secondo due dimensioni:**
  - dimensione temporale: evoluzione dinamica del processo in termini di cicli, fasi, iterazioni, milestones
  - dimensioni delle componenti del processo: attività elaborati, ruoli e risorse umane.

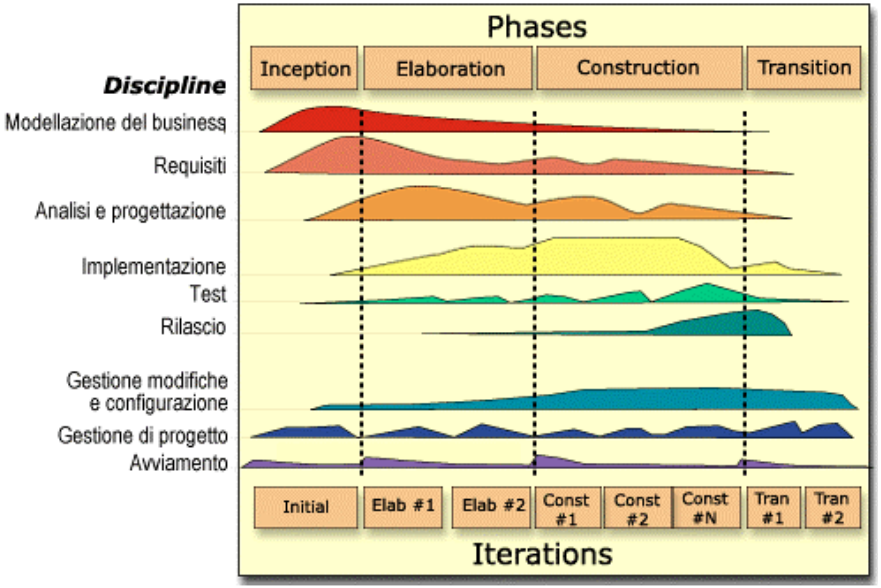
Unified Process

- FASI
  - **Inception**
    - **definizione della fattibilità e della portata del progetto**
  - **Elaboration**
    - **pianificazione attività e risorse**
    - **definizione delle funzionalità**
    - **progettazione architettura**
  - **Construction**
    - **costruzione del prodotto ed evoluzione dell'architettura e dei piani sino al rilascio**
  - **Transition**
    - **correzione dei difetti**
    - **consegna e formazione**
    - **supporto e manutenzione**



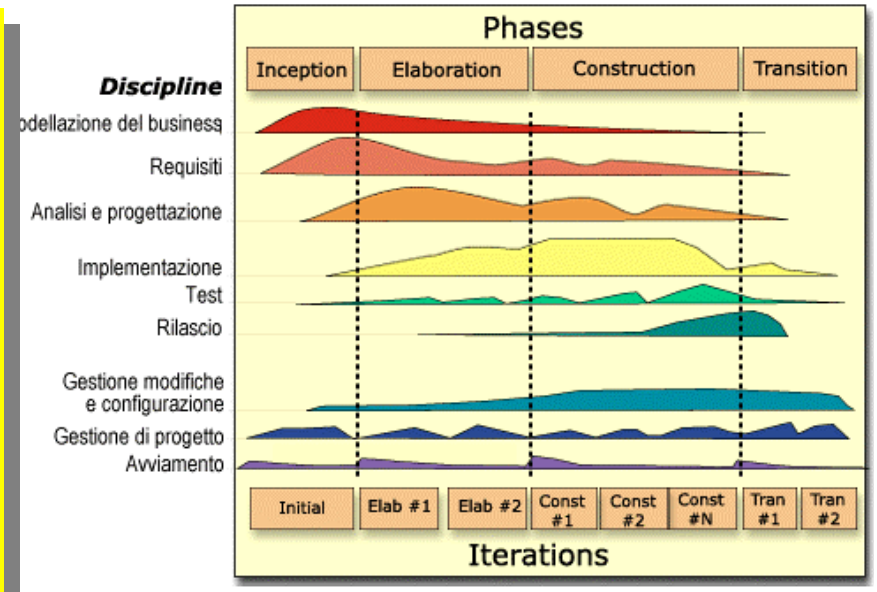
Unified Process

- WORKFLOW principali
  - **Modellazione del business**
    - **comprensione organizzazione**
    - **definizione requisiti di sistema**
  - **Requisiti**
    - **raffinamento e formalizzazione dei requisiti**
    - **modello dei casi d'uso**
  - **Analisi e progettazione**
    - **formalizzazione tecnica dei requisiti**
    - **formalizzazione architettura tecnologica**
  - **Implementazione**
    - **costruzione del software**
  - **Test**
    - **verifica del livello di soddisfacimento dei requisiti**
  - **Rilascio agli utenti**
    - **packaging e distribuzione**



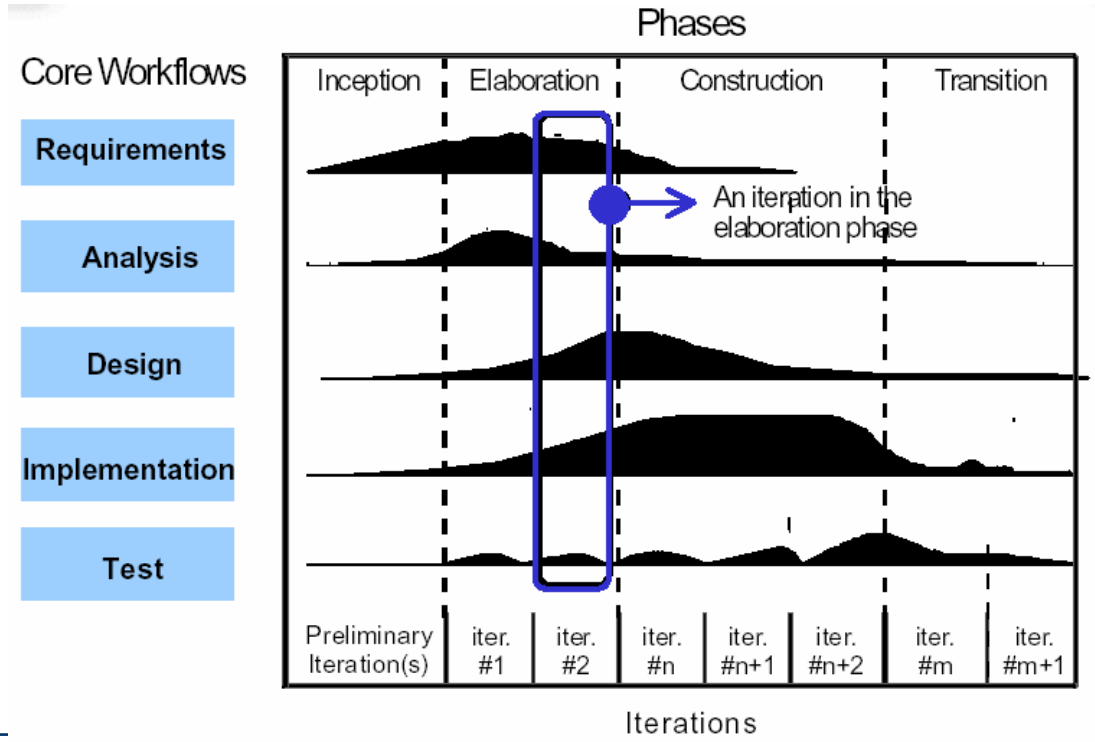
Unified Process

- WORKFLOW di supporto
  - **Gestione di progetto**
    - **gestione dei rischi**
    - **pianificazione del progetto iterativo e della singola iterazione**
    - **controllo evoluzione del progetto iterativo**
  - **Gestione modifiche e configurazione**
    - **gestione delle richieste di modifica**
    - **gestione configurazione del prodotto**
    - **stato e metriche per il controllo del progetto**
  - **Gestione avviamento**
    - **installazione del sw**
    - **Formazione e supporto agli utenti**



Unified Process

- Il processo viene suddiviso in iterazioni, ciascuno dei quali dà luogo ad un prodotto eseguibile (sottoinsieme del prodotto finale)
- Di iterazione in iterazione si giunge al prodotto finale



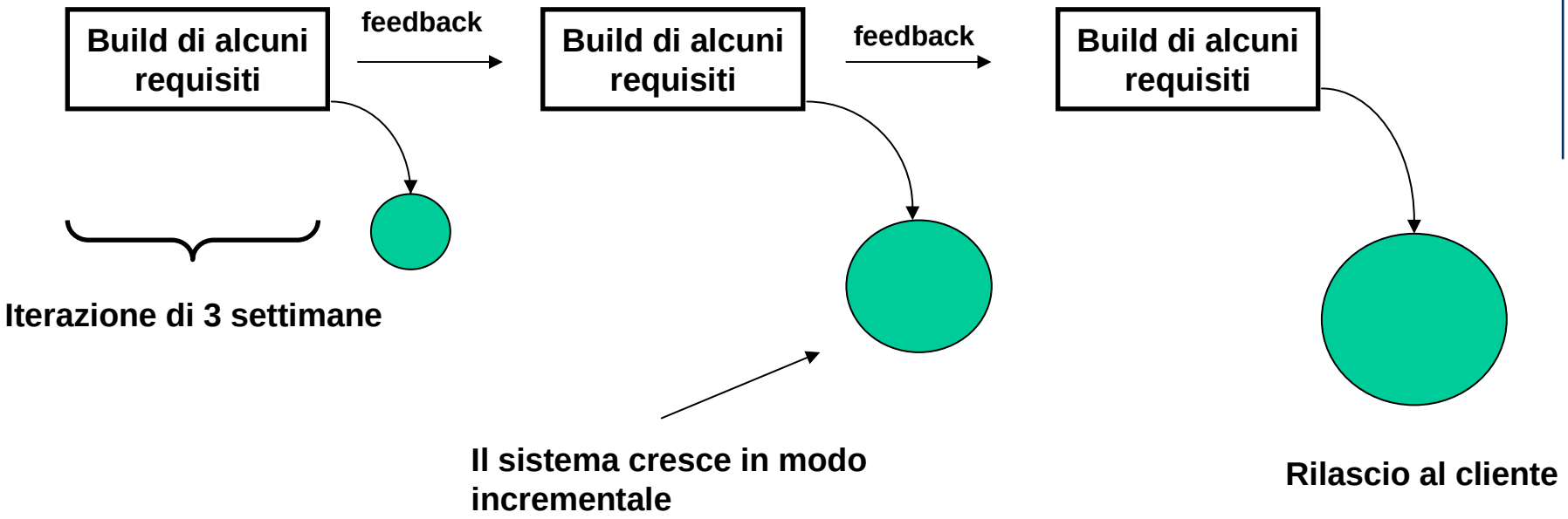
Iterazioni tipiche per progetti medio-piccoli (fonte: Engineering)

Fase	*N. Iterazioni*
INCEPTION	1
ELABORATION	2
CONSTRUCTION	2
TRANSITION	1

Modello evolutivo/iterativo

- **Il processo è composto da diversi cicli in sequenza.**
- **Ogni iterazione è un mini-progetto, il cui risultato è una *release*:**
  - **una parte stabile dell'intero sistema, integrata e testata, che sia stata rilasciata al cliente o no.**

Iterazioni



fonte: Craig Larman

- Lo sviluppo avviene in modo incrementale
- Lunghezza iterazioni: da 1 a 6 settimane

Scelta delle iterazioni

Combinazione di approccio Risk-driven e Client-driven

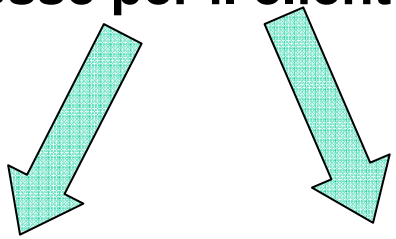
❑ Risk-driven

➢ **Sceglie gli aspetti più rischiosi, più difficili per le prime iterazioni (soprattutto dal punto di vista architetturale**

❑ Client-driven

➢ **La scelta delle features da sviluppare nelle prime iterazioni è fatta dal cliente che sceglie le caratteristiche di maggior valore per lui**

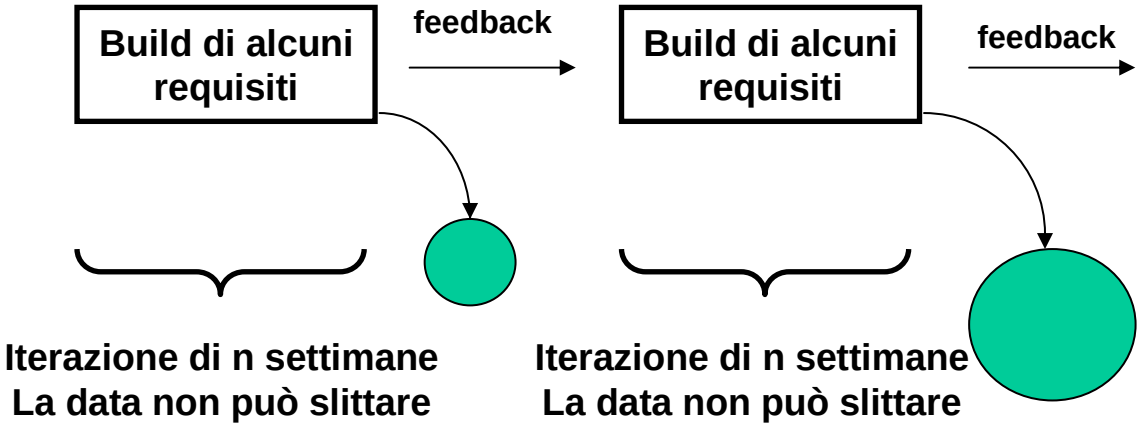
Scelta nelle prime iterazioni delle caratteristiche tecnicamente più rilevanti e difficili (di interesse per chi sviluppa) e di maggior valore di business (di interesse per il cliente)

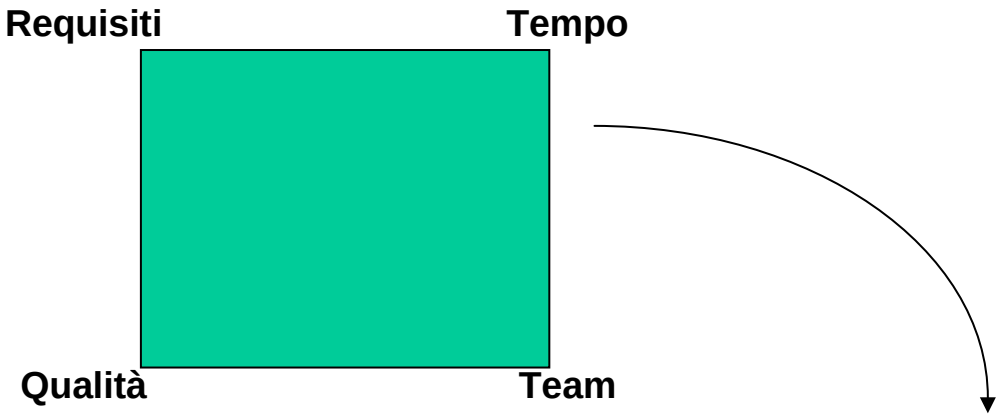


+ QUALITA'

+ VALORE

Sviluppo timeboxed

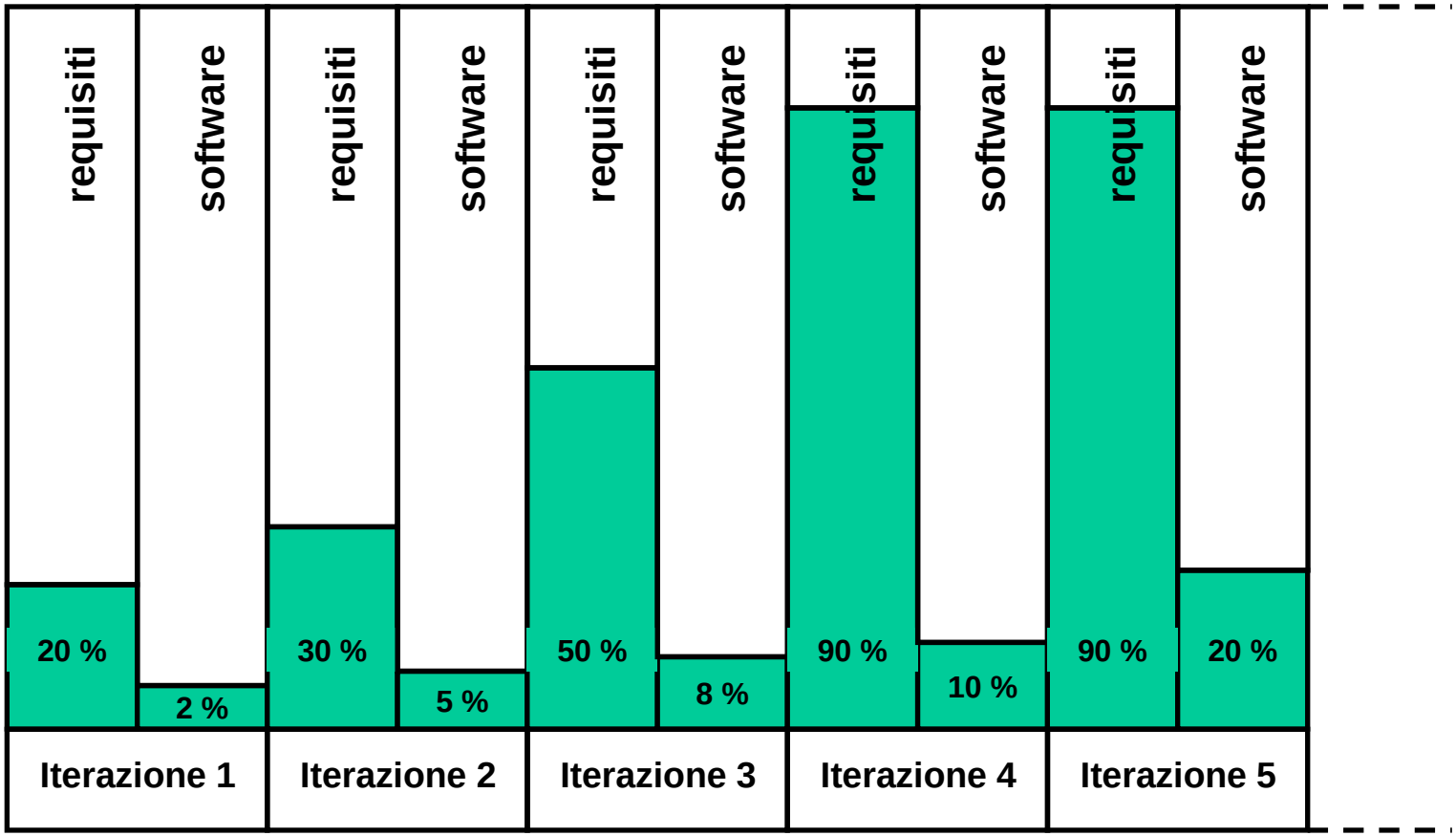




fonte: Craig Larman

- Le 4 variabili: requisiti, tempo, team e qualità
- Lo sviluppo timeboxed elimina la variabile tempo

Analisi dei requisiti



Esempio: dopo le prime 5 iterazioni è sviluppato il 90% dei requisiti e solo il 20% del software

fonte: Craig Larman

Modalità di rilascio

Rilascio incrementale:

rilascio in produzione di un sistema come sequenza (iterazioni) di aumento di caratteristiche

Rilascio evolutivo:

rilascio incrementale con particolare attenzione ad utilizzare il feedback conseguente ad un rilascio come guida per il successivo

L'errore principale (*worst practice*) è:

implementare un approccio evolutivo/iterativo con la "mentalità" waterfall

Modello evolutivo/iterativo

Precisazioni (riassunto):

Iterativo: diversi cicli di sviluppo (iterazioni) ciascuno composto da un ciclo completo (analisi-sviluppo-test o rilascio)

Incrementale: il sistema cresce progressivamente nelle caratteristiche (iterazione dopo iterazione)

Evolutivo: sviluppo incrementale con particolare attenzione al feedback conseguente al consolidamento di ogni iterazione per lo sviluppo della successiva

Adattivo: lo sviluppo si adegua in conseguenza del feedback proveniente dal lavoro precedente (maggiore attenzione all'evoluzione dell'adeguamento al feedback)

Obiettivo (in sintesi):

Rispettare tempi e costi (più che i requisiti!)
Sviluppare e rilasciare ciò che realmente serve (regola di Pareto)
Aumentare la qualità della soluzione

Approcci agili

- **Nascono in alternativa alle metodologie tradizionali e si propongono come metodologie "leggere" caratterizzate dall'essere modelli:**
  - **Adattivi più che predittivi**
    - Non cercano di programmare lo sviluppo nel dettaglio e in modo da soddisfare tutte le specifiche, ma progettano programmi pensati per cambiare nel tempo
  - **People-oriented anziché process-oriented**
    - L'approccio prevede di adattare il processo di sviluppo alla natura dell'uomo... lo sviluppo software deve diventare un'attività piacevole per chi lo opera.
- **Fanno largo uso di best practices:**
  - Consuetudini affermate nello sviluppo del software che, anche se con caratteristiche diverse e intensità diverse, vengono adottate in tutti i processi di sviluppo

Agile Metodologies

AM è una metodologia pratica per la modellazione efficace di sistemi software

- **Una raccolta di practices, guidate da pricipi e valori**
- **Non è un processo formato da prescrizioni**
- **Non fornisce procedure dettagliate su come creare un modello**
- **Fornisce suggerimenti su come essere efficaci modellatori**

Manifesto for Agile SwD

Manifesto for Agile Software Development

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on
the right, we value the items on the left more.

agilemanifesto.org

AM: il Problema

- **Ogni problema è diverso da un altro e richiede diverse strategie.**

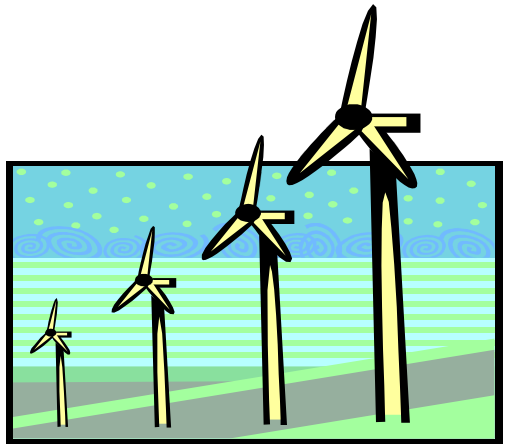
I generali che conducono le guerre pianificano molto, ma sanno che la pianificazione è solo l'inizio: sono ancora più importanti la verifica delle difese del nemico (*creazione del cambiamento*) e la risposta alle azioni del nemico (*risposta al cambiamento*); il successo dipende dalla vittoria sul nemico (*la mission*), non dalla conformità al piano. Non è possibile immaginare un generale che dica: "abbiamo perso la battaglia, ma abbiamo avuto successo perché abbiamo seguito il nostro piano alla lettera". (*da Jim Highsmith*).

- **Esistono progetti (extreme projects) che operano in un contesto simile a campo di battaglia:**
  - ✓ **la mission non è sufficientemente chiara**
  - ✓ **i requisiti sono instabili e variabili**

- **Tali progetti non si adattano ad essere sviluppati con rigorosi metodi *plan-driven***

AM: campo di applicazione

- **Esplorazione petrolifera**
- **Estrazione petrolifera**
  - **mission differente**
  - **gestione differente**
  - **valutazione differente**



	Tecnologia			
Requisiti	**Innovativa, non nota**	**Innovativa, nota**	**Familiare**	**Ben conosciuta**
Irregolari	10	8	7	7
Instabili	8	7	6	5
Normali	7	6	4	3
Stabili	7	5	3	1

Fonte: Jim Highsmith

AM: obiettivi

- Definire e mostrare come mettere in pratica una raccolta di valori, principi e practices pertinenti ad una modellazione efficace e leggera
- Esplorare come applicare tecniche di modellazione ai progetti software, usando un approccio agile come XP o SCRUM
- Esplorare come sia possibile migliorare la modellazione di processi formati da prescrizioni (UP)

Un modello agile è un modello che è buono quanto basta

AM: caratteristiche

- Gli approcci agili sono approcci allo sviluppo:
  - evolutivi ed iterativi
  - timeboxed
  - favoriscono consegne evolutive
  - includono valori e principi che incoraggiano l'agilità: una risposta ai cambiamenti rapida e flessibile
- Rappresentano un sotto-insieme degli approcci evolutivi/iterativi

Abbraccia il cambiamento
(dal titolo del primo testo su XP di Kent Beck)

AM: Valori

- Comunicazione
  - ✓ Tutte le attività prevedono comunicazione (es.: pair programming)
- Coraggio
  - ✓ Misure drastiche quando necessario (es.: eliminare parti di codice)
- Reazione (*feedback*)
  - ✓ feedback immediato da parte di programmatori, clienti, …
- Semplicità
  - ✓ Lavorare nel modo più semplice
- Umiltà

Ottenere un'efficace comunicazione fra tutti gli attori del progetto, sviluppare la soluzione più semplice possibile che soddisfi tutti i bisogni, ottenere ogni feedback sempre e presto su quanto svolto, avere il coraggio di prendere e perseguire le decisioni, avere l'umiltà di ammettere che non si conosce tutto, ma che gli altri possono aggiungere valore alle proprie attività.

Avrei voluto scrivere una lettera più breve,
ma non ne ho avuto il tempo. ***Mark Twain***

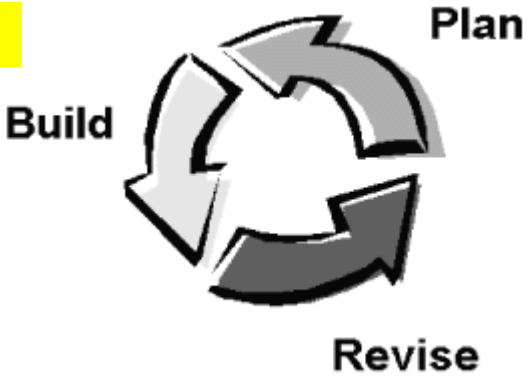
AM: Principi

- **Assumete la semplicità**
  - **Ogni problema va trattato come se si potesse risolvere nel modo più semplice possibile**
- **Accettate e gestite i cambiamenti**
  - **Bisogna conservare la maggior parte delle opzioni mentre si risolvono i problemi più pressanti**
- **Feedback immediati**
  - **Ogni nuova scoperta su come migliorare il design, l'implementazione o il test del sistema deve essere diffusa immdiatamente**
- **Cambiamenti incrementali**
  - **Ogni problema va trattato attraverso una serie di piccole modifiche incrementali**
- **Lavoro di qualità**
  - **Se ti piace il lavoro che fai, produrrai buon software**
- **Permettere il passo successivo l'obiettivo secondario**
- **Massimizzare gli investimenti dei committenti**
- **Modellare con uno scopo**
- **Modelli multipli**
- **Il software è l'obiettivo principale**
- **Viaggiare leggeri**

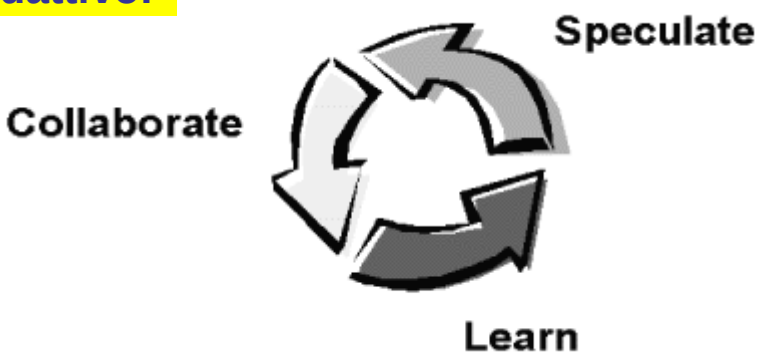
AM: il Modello

Modello lineare:

Modello evolutivo:



Modello adattivo:



J. Highsmith: Adaptive Software Development

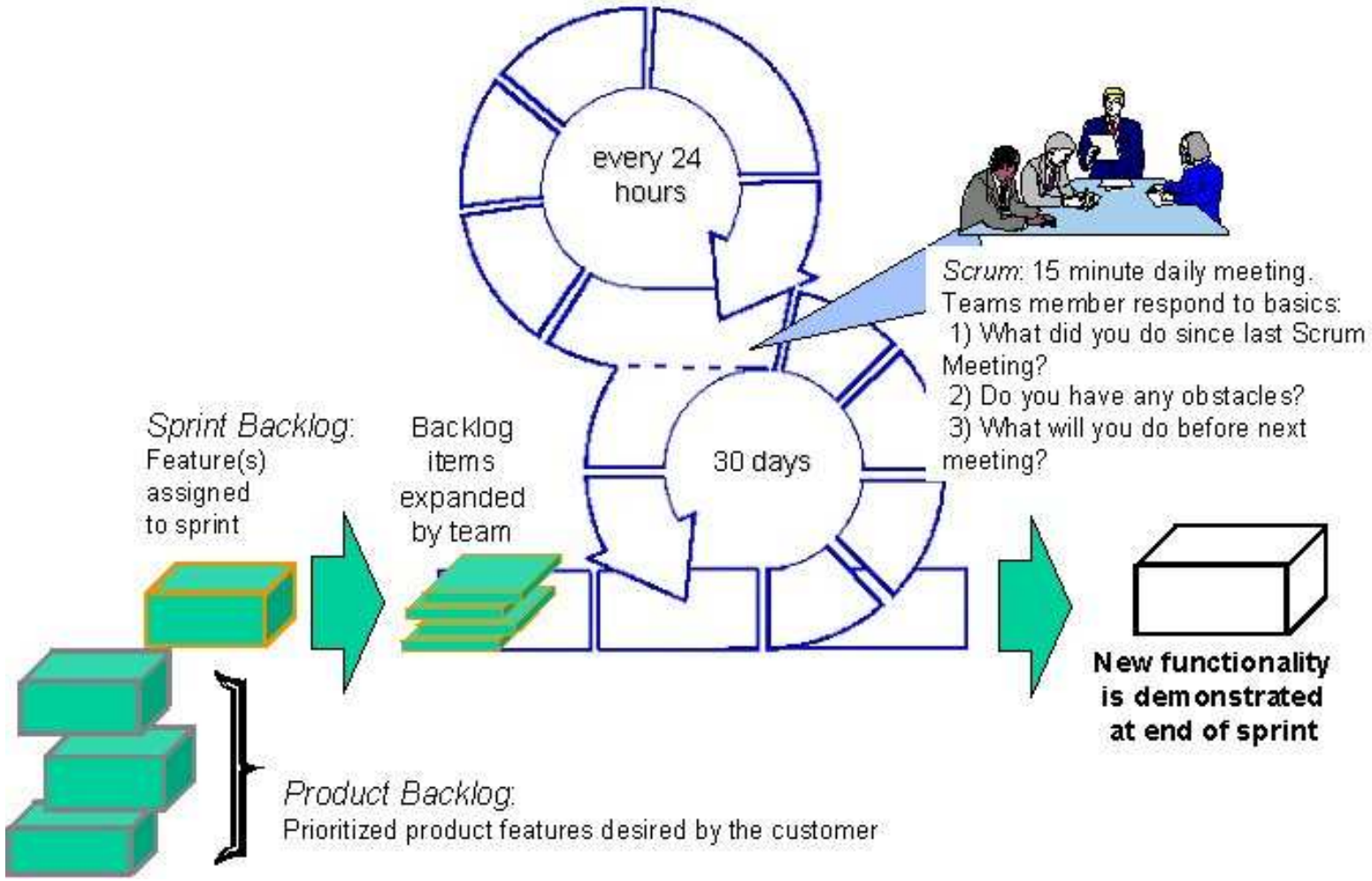
AM: SCRUM

- **SCRUM è un processo "agile" il cui nome deriva dalla terminologia del gioco del Rugby**
- **Si basa sulla teoria della complessità (www.controlchaos.com)**
- **E' un processo**
  - Che può essere adottato per gestire e controllare lo sviluppo del software
  - E' iterativo, incrementale, per lo sviluppo e gestione di ogni tipologia di prodotto
  - Fornisce alla fine di ogni iterazione un set di funzionalità potenzialmente rilasciabili
- **Tre fasi**
  - Pre-game phase
  - Development (Game) phase
  - Post-game phase

AM: SCRUM

- **Pre-game phase**
  - **Planning sub-phase**
    - ✓ Include la definizione del sistema che deve essere sviluppato. Viene creata una Product Backlog List, che contiene tutti i requisiti attualmente conosciuti
  - **Architecture sub-phase**
    - ✓ Viene pianificato un design di alto livello del sistema, inclusa l'architettura, in base agli elementi contenuti nel Product Backlog
- **Development (Game) phase (parte "agile" dell'approccio Scrum)**
  - **Nella Development Phase, il sistema viene sviluppato attraverso una serie di Sprint**
    - ✓ Cicli iterativi nei quali vengono sviluppate o migliorate una serie di funzionalità
    - ✓ Ciascuno Sprint include le tradizionali fasi di sviluppo del software
    - ✓ L'architettura ed il design del sistema evolve durante lo sviluppo negli Sprint
    - ✓ Uno Sprint si svolge in un intervallo di tempo che va da una settimana ad un mese
- **Post-game phase (contiene la chiusura definitiva della release)**

AM: SCRUM



AM: XP

- **L'eXtreme Programming**, o XP, è la metodologia agile che ha ottenuto e sta tuttora ottenendo la maggiore attenzione ed è di fatto la più popolare, il che è attestato dalla numerosa documentazione disponibile.

> " Comunicate con il cliente ed i vostri colleghi, usate semplicità nello sviluppo, testate il software continuamente per verificare cosa funziona e cosa no; molti dei processi "pesanti" sono basati sulla paura: siate invece coraggiosi "

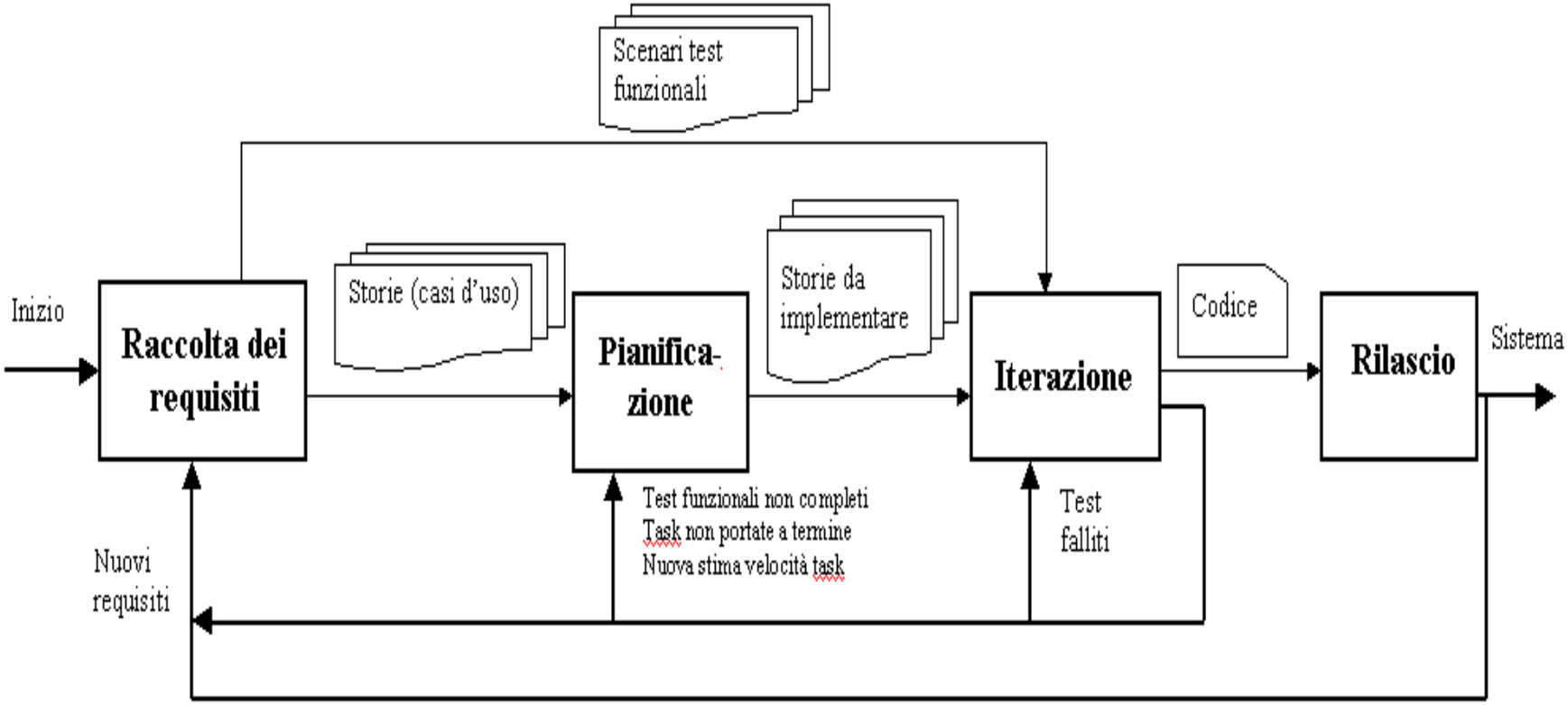
- Tutto il design è incentrato sull'iterazione attuale e non ci si concentra sulle necessità future
- Il risultato è un processo di design che combina disciplina e adattività.
- Il processo di sviluppo di XP è evolutivo e iterativo e si basa sul refactoring ad ogni iterazione di un semplice sistema base

AM: practices XP

- **Planning game**
  Determina l'obiettivo e i tempi della prossima release
- **Small releases**
  Mettere velocemente in produzione un piccolo sistema
- **Metaphor**
  Guida lo sviluppo attraverso una semplice "storia" condivisa che descrive l'intero sistema
- **Simple design**
  Il sistema è concepito nel modo più semplice possibile
- **Testing**
  Deve essere effettuato costantemente durante lo sviluppo (test-first-development)
- **Refactoring**
  Ristrutturare il sistema senza cambiarne il comportamento
- **Pair programming**
  Tutto il codice è scritto da due programmatori che lavorano in coppia alla stessa macchina
- **Collective ownership**
  Chiunque può cambiare qualsiasi parte del codice quando vuole
- **Continuous integration**
  Integrazione e build del sistema vengono effettuate più volte al giorno
- **40-hour week**
  E' sconsigliato lavorare più di 40 ore la settimana
- **On-site customer**
  Un rappresentate del cliente dev'essere sempre a disposizione
- **Coding standards**
  Supportano la comunicazione durante la produzione del codice

AM: processo XP

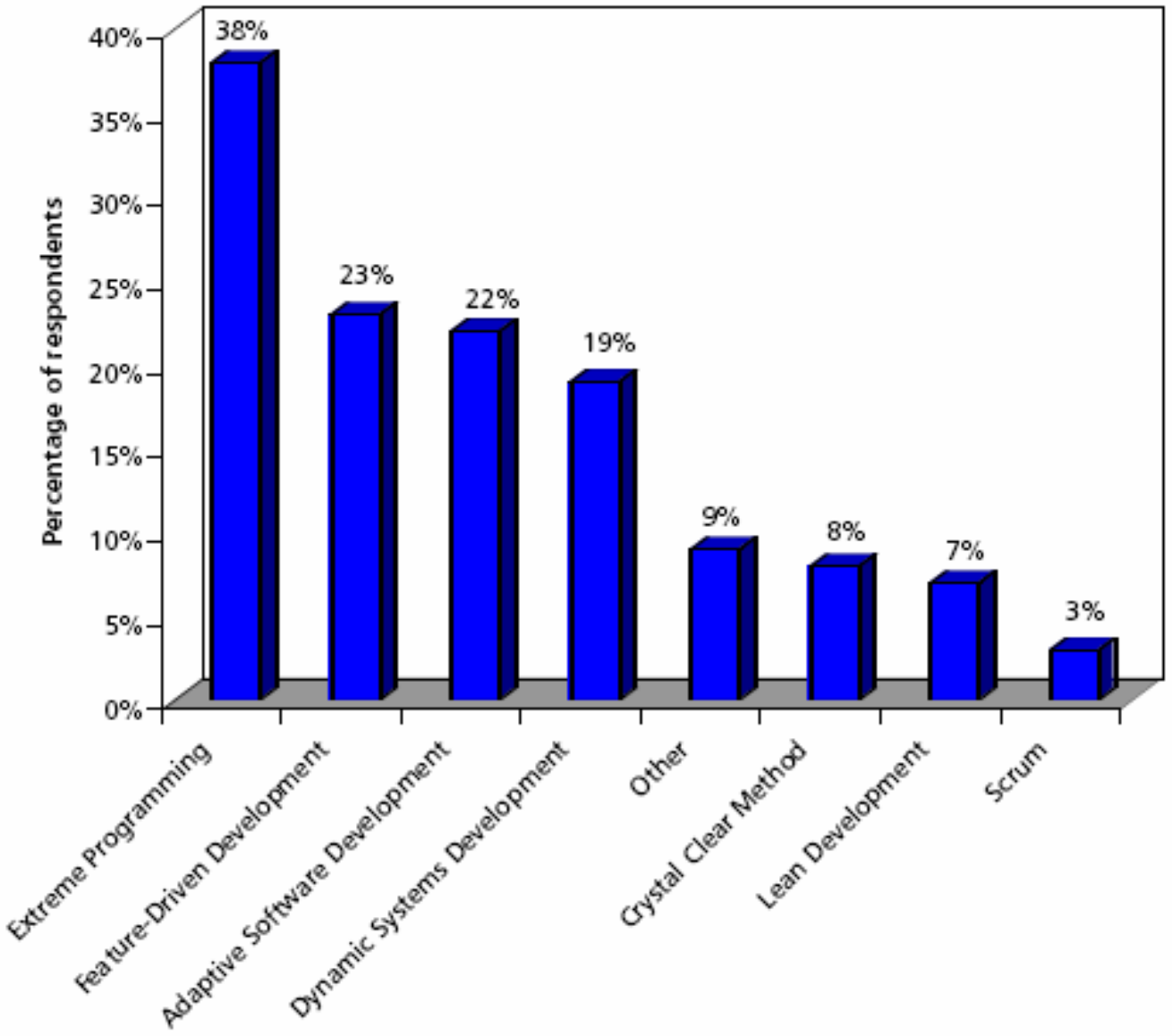
Il processo dell'XP



AM: diversi approcci

The Cutter survey size involving agile versus heavy methodologies involved 200 IS/IT managers, with about 33% of the respondents coming from North America; 20% from Europe; 10% from Australia; 8% from India; 8% from Asia; and the remainder spread across South America, Africa, and the Middle East.

http://www.cutter.com/articles (2001)



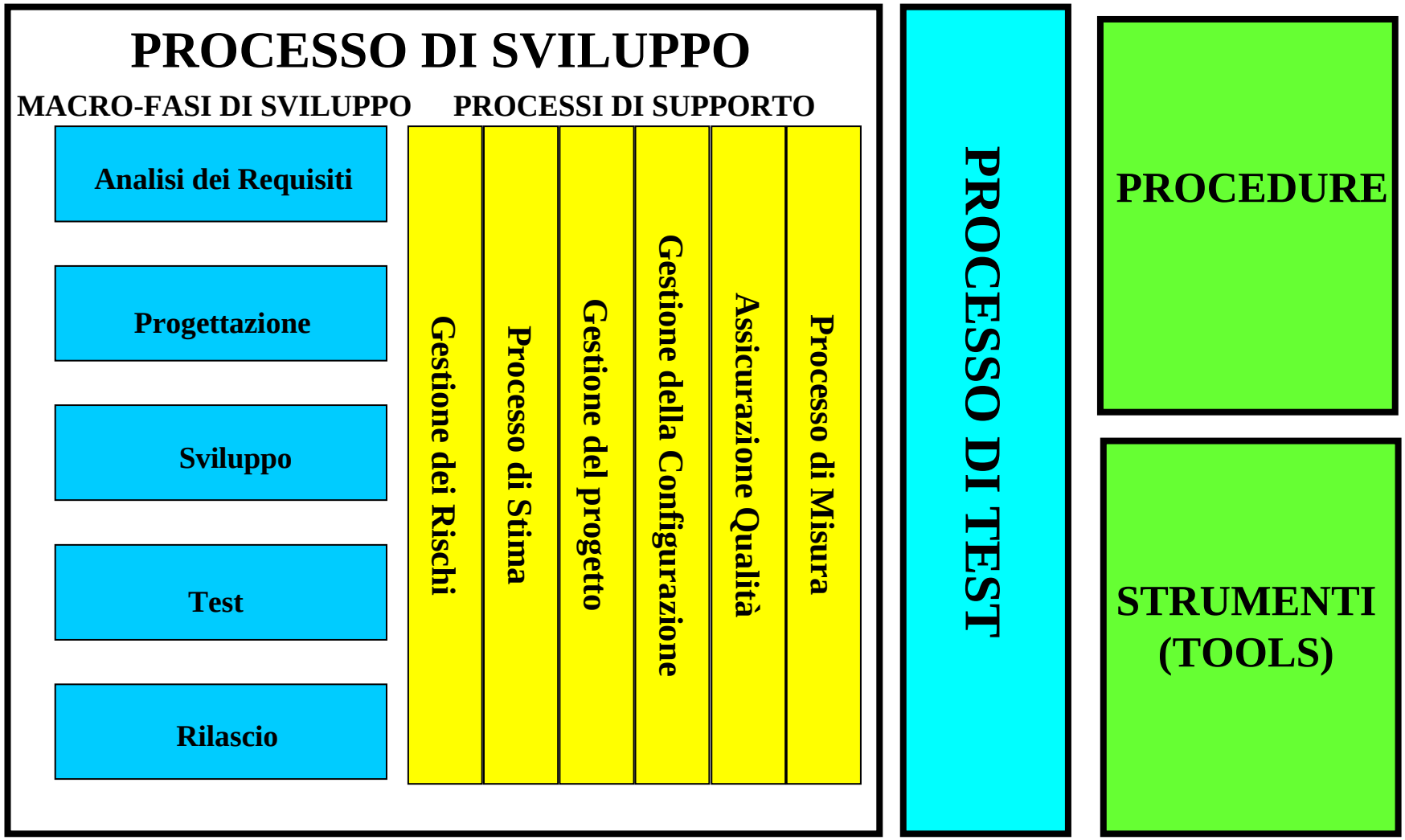
Matrice di confronto

	Build & Fix	Waterfall	Iterativo	Evolutivo	Agile
Gestione del rischio	NO	NO	SI	SI	NO
Definizione iniziale tutti requisiti	SI	SI	PARZ.	PARZ.	PARZ.
Complessità progetto	Bassa	Bassa	Media	Media	Alta
Dinamica dei requisiti	Bassa	Bassa	Media	Alta	Alta
Tecnologie e competenze	Tradiz.	Tradiz.	Nuove	Nuove	Nuove
Ripetibilità del processo	NO	SI	SI	SI	SI

Forza/debolezza

	PUNTI DI FORTI	PUNTI DEBOLI
Build & Fix	Adatto a progetti piccoli	Non adatto a progetti complessi; non ripetibile; legato alla specifica competenza
Waterfall	Comprensibile, controllabile, adeguato ai contratti, chiare milestones	Difficile gestione requisiti e rapporto con cliente, ritarda e rimanda l'individuazione dei problemi
Iterativo	Produce in anticipo prime versioni del prodotto	Necessita di capacità nella pianificazione e gestione delle iterazioni
Evolutivo	Consente di utilizzare i feedback, migliore capacità di gestire la variabilità dei requisiti	Necessità di skill e preparazione adeguati sia nello sviluppo, che nella gestione (es.: dei rischi)
Agile	Adatto a progetti particolarmente innovativi e con requisiti non ben definiti	Necessita di grande disciplina e motivazione; non facilmente comprensibile dal cliente, generalmete non adeguato ai contratti

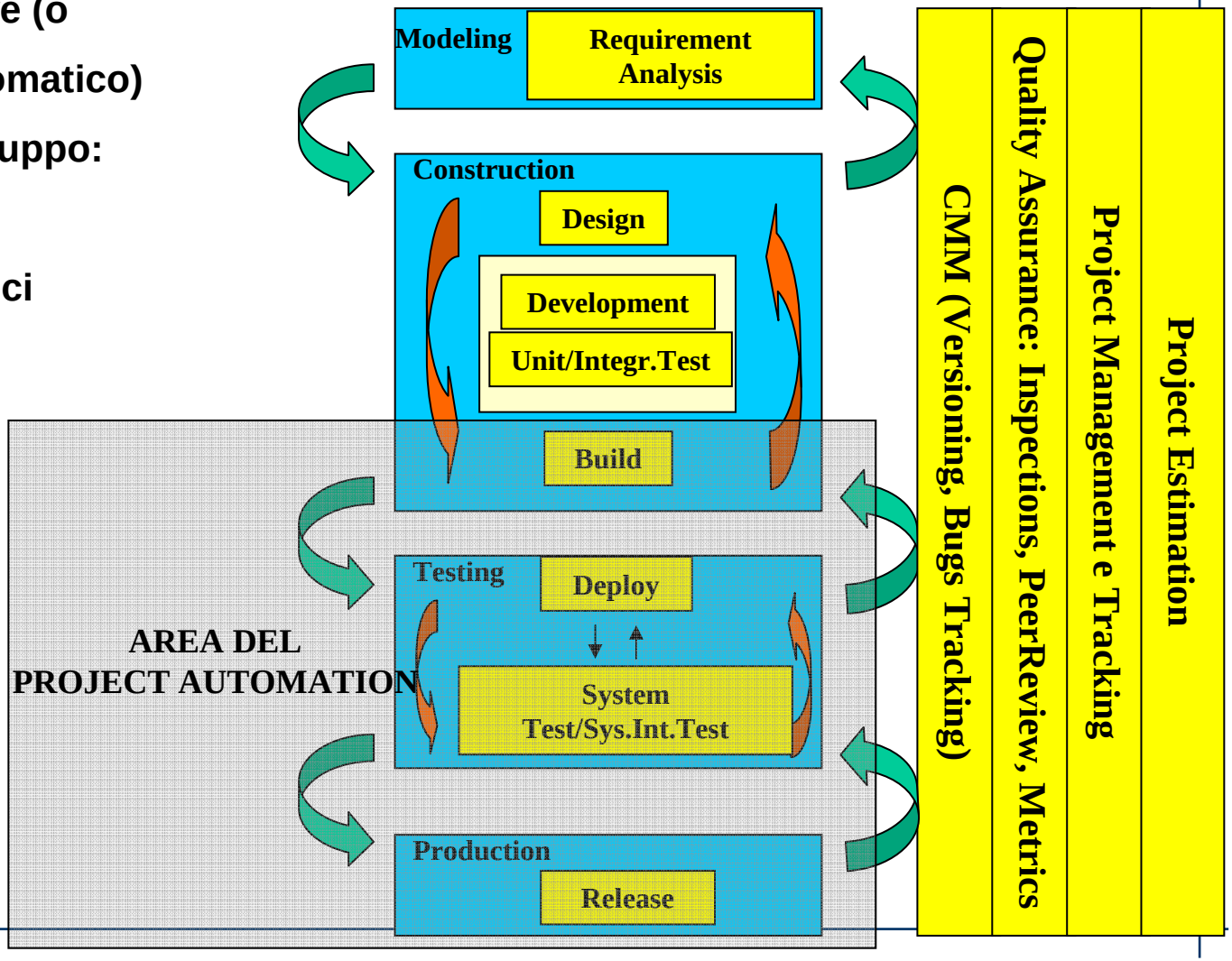
Processo di sviluppo



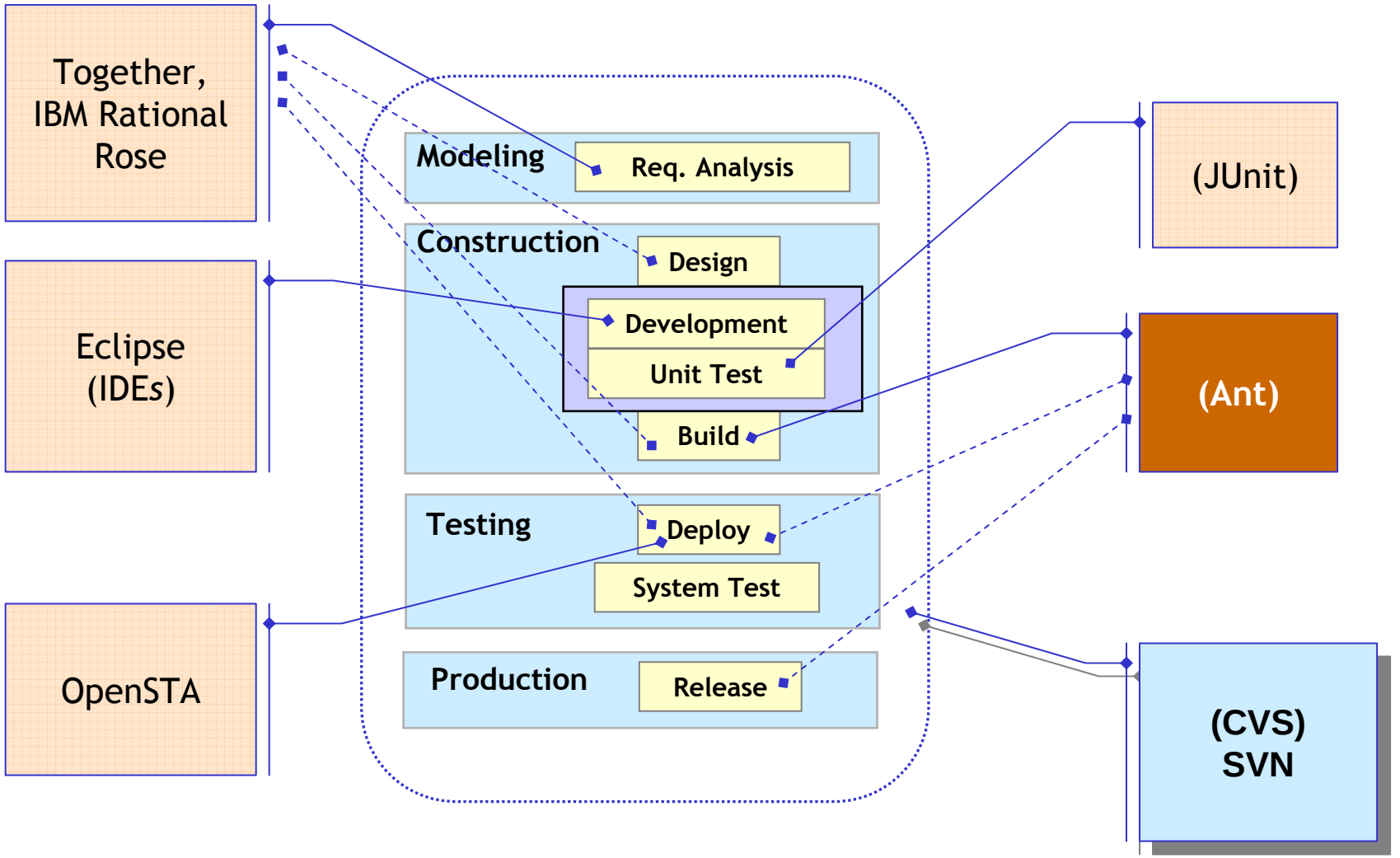
Project Automation

Esecuzione automatica delle fasi critiche e ripetitive (o ripetibili in modo automatico) di un processo di sviluppo:

- **Build Process**
- **Unit Test Automatici**
- **Deploy**
- **System Test (TFA)**
- **Release**



Project Automation



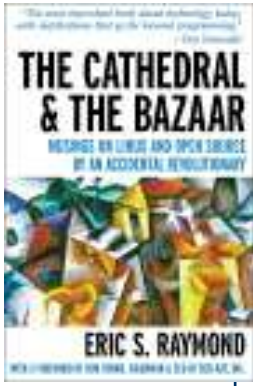
FOSS: organizzazione

Rimasi non poco sorpreso dallo stile di sviluppo di Linus Torvalds – diffondere le release presto e spesso, delegare ad altri tutto il possibile, essere aperti fino alla promiscuità. Nessuna cattedrale da costruire in silenzio e reverenza. Piuttosto, la comunità Linux assomigliava a un grande e confusionario bazaar, pullulante di progetti e approcci tra loro diversi (efficacemente simbolizzati dai siti contenenti l'archivio di Linux dove apparivano materiali prodotti da chiunque). Un bazaar dal quale soltanto una serie di miracoli avrebbe potuto far emergere un sistema stabile e coerente.

E Raymond, La cattedrale e il bazaar, 1998.

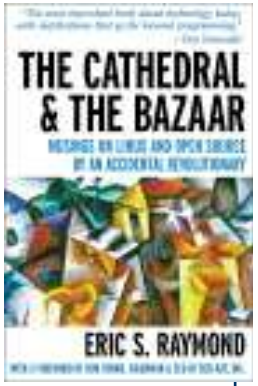
Lezioni di Raymond

1. **Ogni buon lavoro software inizia dalla frenesia personale di uno sviluppatore.**
2. **I bravi programmatori sanno cosa scrivere. I migliori sanno cosa riscrivere (e riusare).**
3. **“Preparati a buttarne via uno, dovrai farlo comunque” (per arrivare alla soluzione, preparati a ricominciare almeno una volta).**
4. **Se hai l’atteggiamento giusto, saranno i problemi interessanti a trovare te.**
5. **Quando hai perso interesse in un programma, l’ultimo tuo dovere è passarlo ad un successore competente.**
6. **Trattare gli utenti come co-sviluppatori è la strada migliore per ottenere rapidi miglioramenti del codice e debugging efficace**



Lezioni di Raymond

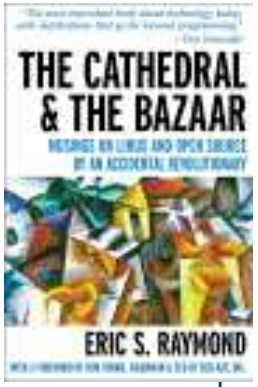
7. **Distribuisci presto. Distribuisci spesso. E presta ascolto agli utenti.**
8. **Stabilita una base di beta-tester e co-sviluppatori sufficientemente ampia, ogni problema verrà rapidamente definito e qualcuno troverà la soluzione adeguata.**
9. **Meglio combinare una struttura dati intelligente e un codice non eccezionale che il contrario.**
10. **Se tratti i beta-tester come se fossero la risorsa più preziosa, replicheranno trasformandosi davvero nella risorsa più preziosa a disposizione.**
11. **La cosa migliore, dopo l'avere buone idee, è riconoscere quelle che arrivano dagli utenti. Qualche volta sono le migliori.**
12. **Spesso le soluzioni più interessanti e innovative arrivano dal fatto di esserti reso conto come la tua concezione del problema fosse errata.**



Lezioni di Raymond

13. La perfezione (nel design) si ottiene non quando non c'è niente altro da aggiungere, bensì quando non c'è più niente da togliere.

14. Ogni strumento dovrebbe rivelarsi utile nella maniera che ci si attende, ma uno strumento davvero ben fatto si presta ad utilizzi che non ci si aspetterebbe mai.

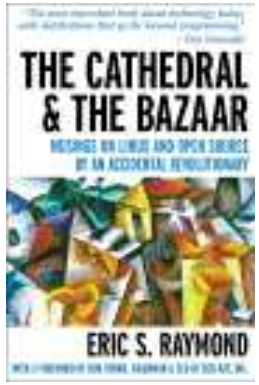


15. Quando si scrive del software per qualunque tipo di gateway, ci si assicuri di disturbare il meno possibile il flusso dei dati – e *mai* buttare via alcun dato a meno che il destinatario non ti ci costringa.

16. Quando il linguaggio usato non è affatto vicino alla completezza di Turing, un po' di zucchero sintattico può esserti d'aiuto.

Lezioni di Raymond

17. **Un sistema di sicurezza è sicuro finché è segreto. Meglio diffidare degli pseudo-segreti.**
18. **Per risolvere un problema interessante, comincia a trovare un problema che risvegli il tuo interesse.**
19. **Stabilito che il coordinatore dello sviluppo abbia a disposizione un medium almeno altrettanto affidabile di Internet, e che sappia come svolgere il ruolo di leader senza costrizione, molte teste funzionano inevitabilmente meglio di una sola.**



Principi e best practices

- Avere una forte motivazione a trovare la soluzione
- Individuare una soluzione “aperta” e convincente
- Collaborare, condividere, attrarre l’interesse della comunità
- Non partire da zero
- Distribuire presto, distribuire spesso
- Saper riconoscere le buone idee degli altri ed accettarne i contributi
- Ascoltare gli utenti
- Comunicare efficacemente

MS Synch & Stabilize

Modello a spirale:

- continua sincronizzazione e periodica stabilizzazione
- sviluppo in modalità top-down con aggiunta di funzionalità
- sviluppo e test in parallelo

MS Synch & Stabilize

- **planning phase**
  - **visione informale di progetto (input da clienti)**
  - **specifiche (modularità)**
  - **pianificazione (team: pm, developers, testers; buffer time)**
- **development phase**
  - **sottoprogetto I (coding, debugging, testing, daily build)**
  - **sottoprogetto II**
  - **sottoprogetto III**
- **stabilization phase**
  - **virtual freeze**
  - **testing interno**
  - **testing esterno (beta test)**
  - **release finale (gold master copy)**

Open Source

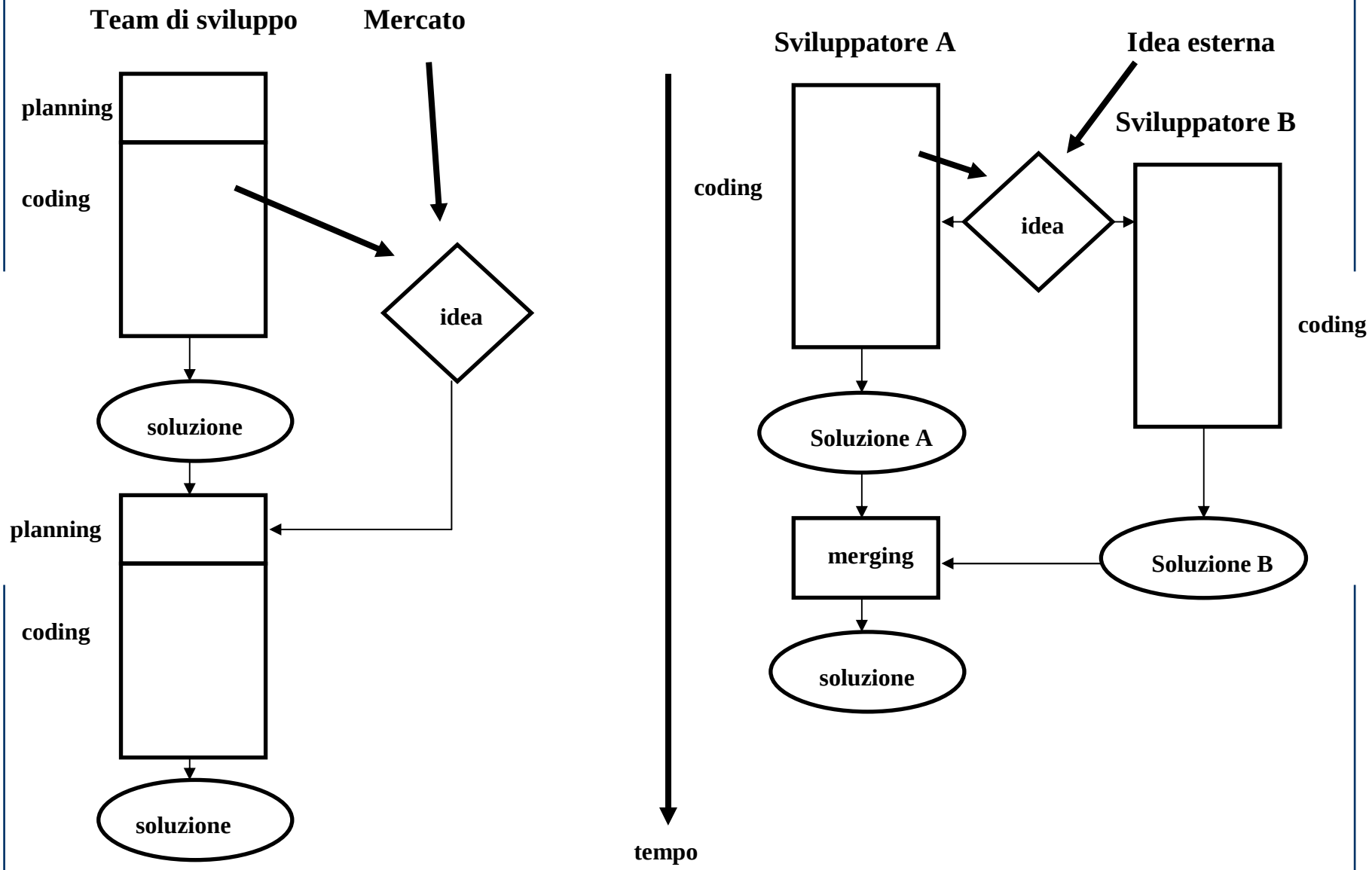
Caratteristiche presenti nei diversi modelli e in OS:

- **Build & Fix**: importanza delle risorse e delle competenze
- **Iterativo**: divisione in sottoprogetti e sviluppo incrementale
- **Evolutivo**: esecuzione parallela e convergenza verso il rilascio, valore al feedback
- **Agile**: risorse e competenze, features, feedback

Open Source - Linux

- **Estesa modularità**
  - **sviluppo parallelo**
  - **sviluppo distribuito**
- **Sviluppo parallelo**
  - **raccolta di nuove idee**
  - **ridondanza**
  - **velocità di sviluppo**
- **Qualità del prodotto**
  - **testing esteso**
  - **disponibilità del codice**
  - **feedback dal campo**

Ms S&S vs OS



Agile vs OS

AGILE over OS

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

usati, ma non diffusi

esiste, ma non diffusa

Servizi, scarse specifiche

obiettivi principali, milestones

Bibliografia

- **Muffatto M., Faldani M., *Open Source – Strategie, organizzazione, prospettive*, Il Mulino, 2004**

- **Raymond E.S., *The Cathedral and the Bazaar*, O'Really, 1999, http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/ (traduzione italiana: http://www.apogeonline.com/openpress/cathedral)**