Università degli Studi di Napoli
"Federico II"

Ingegneria del Software

a.a. 2013/14

Lezione 19: Il processo software

Obiettivi della lezione

- Comprendere il concetto di Processo Software
- Comprendere il concetto di Modello di Processo Software
- Comprendere il Modello di Processo a Cascata

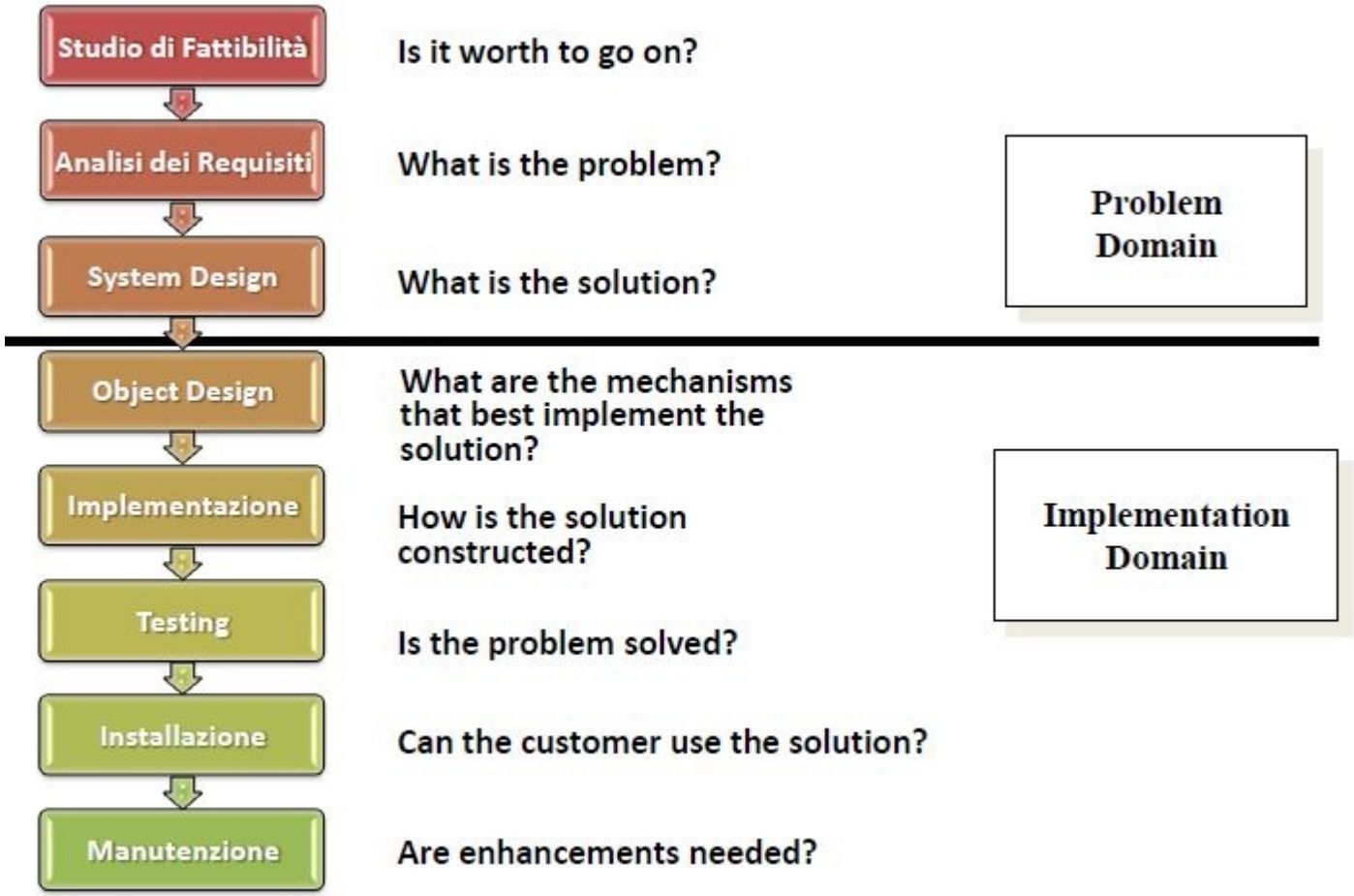
Processo/Ciclo di vita del software

- Insieme organizzato di attività che sovrintendono alla costruzione del software da parte del team di sviluppo utilizzando metodi, tecniche, metodologie e strumenti.
- È suddiviso in fasi, che vanno dalla nascita fino alla dismissione (o morte) del software.
  - Per un parallelo con la biologia, è noto come ***ciclo di vita del software***
- Molti autori usano i termini ***processo*** *[di sviluppo del] software* e ***ciclo di vita*** *del software* come sinonimi.

Processo software: Standard IEEE 610.12-1990

- *"Sofware development process: The process by which user needs are translated into a software product. The process involves translating user needs into software requirements, transforming the software requirements into design, implementing the design in code, testing the code, and sometimes, installing and checking out the software for operational use.*
  - *Note: These activities may overlap or be performed iteratively.*

Attività richieste nel processo di sviluppo software



Attività del processo: fasi alte

- **Studio di fattibilità**: Valutazione preliminare di costi e benefici
- **Raccolta dei requisiti: Raccolta** dei bisogni dell'utente e dominio del problema
- **Analisi dei requisiti: Analisi** completa dei bisogni dell'utente e dominio del problema
- **System Design:** Scomposizione del sistema in componenti e moduli
- **Object Design:** Scelta dei Dettagli interni a ciascuna componente
-

Fasi basse del processo

- **Implementazione**: ogni modulo viene codificato nel linguaggio scelto e testato in isolamento
- **Testing**
  - Composizione dei moduli nel sistema globale
  - Verifica del corretto funzionamento del sistema
  - Validazione che il sistema faccia ciò che vuole il cliente
- **Installazione**: distribuzione e gestione del software presso l'utenza
- **Manutenzione**: evoluzione del SW. Segue le esigenze dell'utenza. Comporta ulteriore sviluppo per cui racchiude in sé nuove iterazioni di tutte le precedenti fasi

Problemi nel processo di sviluppo del software

- Ha aspetti creativi e scarsamente automatizzabili, è basato su aspetti valutativi e decisionali
  - Non è possibile (almeno con i sistemi attuali) automatizzarlo
- I requisiti sono complessi e ambigui
  - Il cliente potrebbe non avere idee chiare, dall'inizio, su cosa deve fare il prodotto
- I requisiti sono variabili
  - Cambiamenti tecnologici, organizzativi, etc…
- Modifiche frequenti sono difficili da gestire
  - Difficile stimare i costi ed identificare cosa consegnare al cliente

I modelli di processo

Cosa sono i Modelli di Processo

- Come si organizza la sequenza delle attività del processo software, per massimizzare alcuni aspetti qualitativi di produzione
- Il processo di sviluppo deve essere **modellato** esplicitamente, per poter essere gestito e monitorato
- I modelli di processo software sono descrizioni precise e formalizzate delle attività, delle trasformazioni, dei deliverable e degli eventi per realizzare e/o ottenere l'evoluzione del software
- Obiettivo:
  - introdurre stabilità, controllo e organizzazione in una attività tendenzialmente destrutturata.

Modelli di processo: caratteristiche (1)

- Una strutturazione dell'organizzazione del lavoro nelle aziende di produzione del software in:
  - fasi della produzione,
  - tipi di attività,
  - collegamento ed interfacciamento,
  - controllo e misura,
- ma anche linee guida per: organizzare, pianificare, dimensionare personale, assegnare budget, schedulare e gestire, …

Modelli di processo: caratteristiche (2)

- definire e prescrivere prodotti e documenti da rilasciare al committente (**deliverable**)
- determinare e classificare metodi e strumenti più adatti a supportare le attività previste
- framework per analizzare, stimare, migliorare ...

Diverse tipologie di Modelli di processo

- Esempi di Modelli di processo
  - Waterfall (cascata)
  - Evolutivo
  - Basato sul riuso
  - eXtreme Programming
  - Test Driven Develpment
- Molti aspetti influenzano la definizione del modello
  - specificità dell'organizzazione produttrice
  - know-how
  - area applicativa e particolare progetto
  - strumenti di supporto

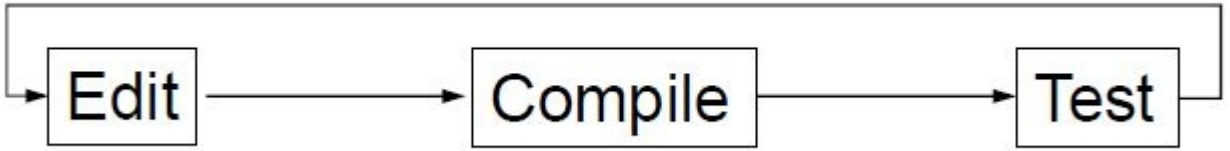
Come valutare un modello di processo? (1)

- Come per il software, esistono dei parametri per valutare la "bontà" di un modello con cui sviluppare il software:
  - **Comprensibilità**: Quanto è facile apprendere il processo?
  - **Visibilità** : Quanto facilmente il processo fa capire a che punto dello sviluppo si è giunti?
  - **Supportabilità** (CASE tools): Esistono tool che supportano il processo? Quali e quante fasi sono supportate? A che livello?
  - **Accettabilità**: Le persone coinvolte nel processo manifestano il loro consenso?

Come valutare un modello di processo? (2)

- **Affidabilità**: Il processo facilità l'individuazione di errori? Il software prodotto è di alta qualità?
- **Robustezza**: Quanto è robusto il processo nel gestire cambiamenti in corso d'opera?
- **Mantenibilità**: Il processo è in grado di adattarsi ai cambiamenti nell'organizzazione che lo usa?
- **Rapidità**: Il porta ad un rapido sviluppo del software?

Un semplice (e sconsigliabile) modello di processo



- Molto veloce, feedback rapido
- Molti strumenti disponibili
- Specializzato per la codifica
- Non incoraggia la produzione di documentazione
- Non è scalabile (in the large, in the many)

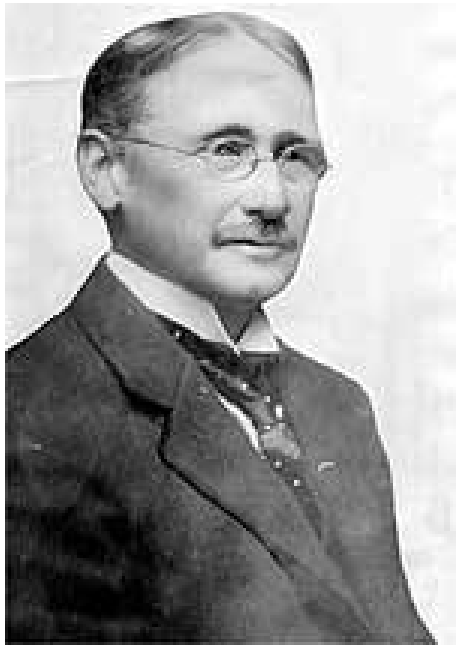
- Difficilmente gestibile durante la fase di manutenzione

Valutazione dell'Edit-Compile-Test

- **Comprensibilità**:
- **Visibilità** :
- **Supportabilità**:
- **Accettabilità**:
- **Affidabilità**:
- **Robustezza**:
- **Mantenibilità**:
- **Rapidità**:

Il modello a cascata

L’approccio Tayloristico

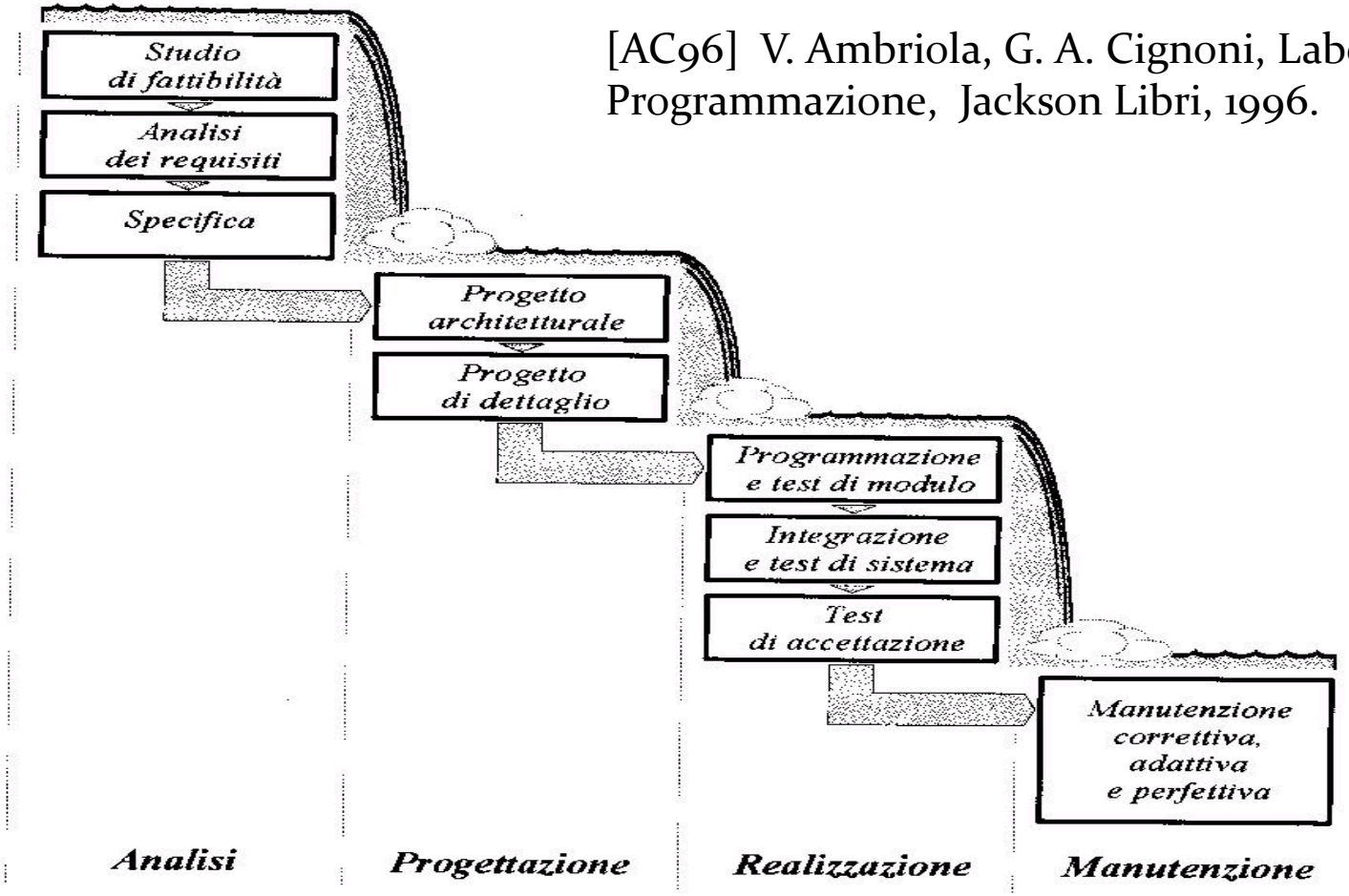


- Frederick Taylor – “The Principles of Scientific Management” (1911)
  - Gestione gerarchica
  - Passi fissi, non fluidi
  - Separare nettamente le postazioni di lavoro
  - Il lavoro, una volta suddiviso, diventa specializzato in un singolo task
  - Processo orientato al prodotto, non al cliente
    - “Any customer can have a car painted any color that he wants so long as it is black” - *Henry Ford*
  - Processo ripetibile

Modelli a Cascata (Waterfall) - ’70

- Applicazione dell’approccio Tayloristico all’informatica
- Popolare negli anni ’70 in reazione al “code and fix” originario
- Modello sequenziale lineare
  - progressione sequenziale (in cascata) di fasi, senza cicli, al fine di meglio controllare tempi e costi
  - definisce e separa le varie fasi e attività del processo
    - nullo (o minimo) overlap fra le fasi
  - uscite intermedie: semilavorati del processo (documentazione di tipo cartaceo, programmi)
    - formalizzati in struttura e contenuti
  - consente un controllo dell’evoluzione del processo.

Modello Waterfall [AC96]



[AC96] V. Ambriola, G. A. Cignoni, Laboratorio di Programmazione, Jackson Libri, 1996.

fasi alte del processo ***fasi basse del processo***

Organizzazione sequenziale delle fasi

- Ogni fase raccoglie un insieme di attività omogenee per metodi, tecnologie, skill del personale, etc.
- Ogni fase è caratterizzata da:
  - Attività (**task**),
  - Prodotti di tali attività (**deliverable**),
  - Controlli di qualità/stato (**quality control measure**)
- La fine di ogni fase è un punto rilevante del processo (**milestone**)
- I semilavorati output di una fase sono input alla fase successiva
- I prodotti di una fase vengono “congelati”, ovvero non sono più modificabili se non innescando un processo formale e sistematico di modifica

I documenti prodotti con il modello Waterfall

Activity	**Output documents**
Requirements analysis	Feasibility study, Outline requirements
Requirements definition	Requirements document
System specification	Functional specification, Acceptance test plan Draft user manual
Architectural design	Architectural specification, System test plan
Interface design	Interface specification, Integration test plan
Detailed design	Design specification, Unit test plan
Coding	Program code
Unit testing	Unit test report
Module testing	Module test report
Integration testing	Integration test report, Final user manual
System testing	System test report
Acceptance testing	Final system plus documentation

Modello a cascata: vantaggi e svantaggi

- **Vantaggi**
  - ha definito molti concetti utili (semilavorati, fasi ecc.)
  - ha rappresentato un punto di partenza importante per lo studio dei processi SW
  - facilmente comprensibile e applicabile
  - E' un modello molto rigoroso: si applica bene in qualunque contesto in cui i requisiti siano stabili e chiari

Modello a cascata: vantaggi e svantaggi

- **Svantaggi**
  - interazione con il committente solo all'inizio e alla fine del processo
    - requisiti congelati alla fine della fase di analisi
    - Rischi legati a requisiti utente spesso imprecisi: "l'utente sa quello che vuole solo quando lo vede"
    - Rischi legati a errori nei requisiti scoperti solo alla fine del processo
  - il prodotto diventa installabile solo quando è totalmente finito
    - né l'utente né il management possono giudicare prima della fine dell'adesione del sistema alle proprie aspettative

Valutazione del modello a cascata

- **Comprensibilità**:
- **Visibilità** :
- **Supportabilità**:
- **Accettabilità**:
- **Affidabilità**:
- **Robustezza**:
- **Mantenibilità**:
- **Rapidità**:

Nella realtà …

- l’applicazione evolve durante tutte le fasi
  - overlap di attività e retroazioni (loop) sono inevitabili!
  - in alcuni casi è auspicabile sviluppare prima una parte del sistema e poi completarlo (utente finale= mercato)
  - … la manutenzione non può essere considerata marginale

Varianti migliorative del modello a cascata

I Prototipi Throw-Away

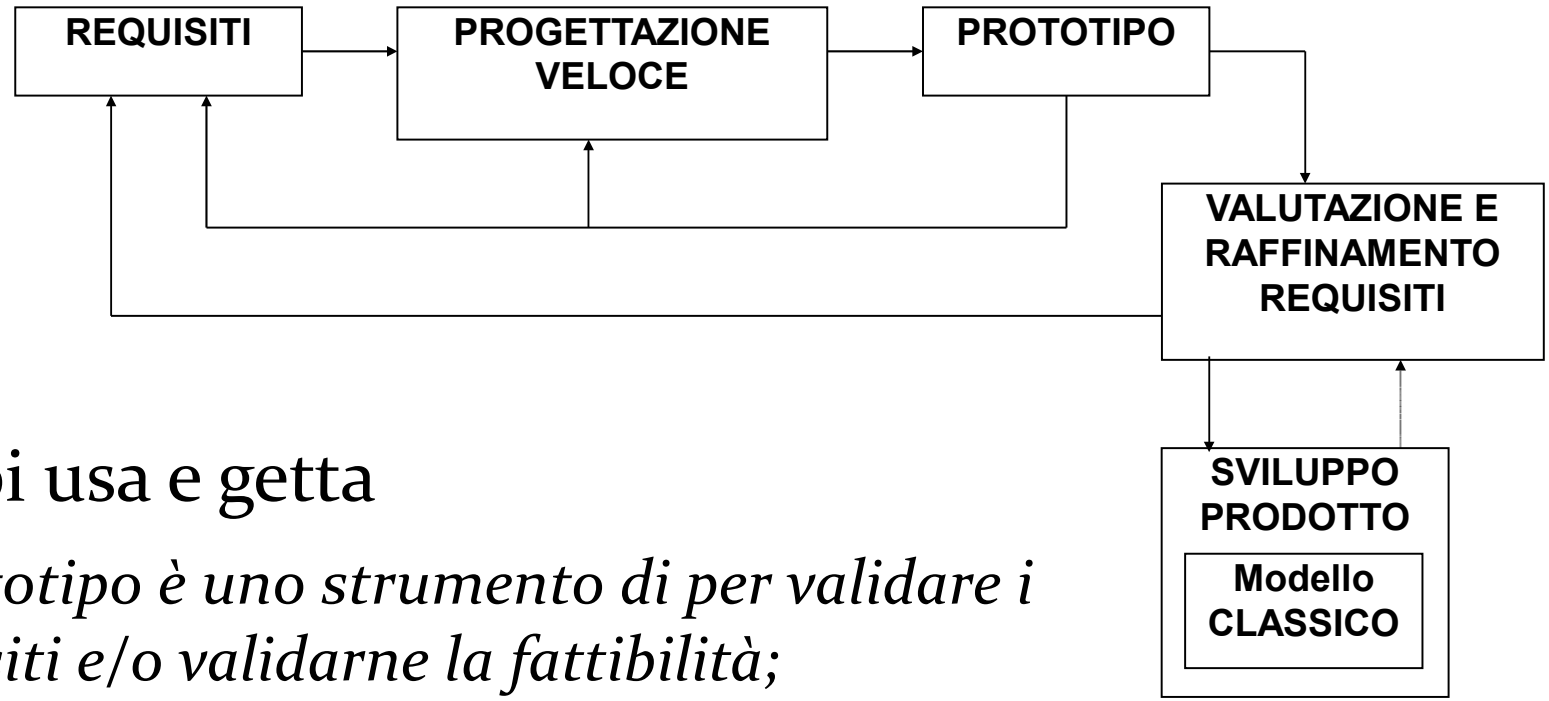
Prototipazione Usa e Getta

- Prototipazione di tipo usa e getta (throw-away)
  - L'obiettivo è capire i requisiti del sistema e quindi sviluppare una definizione migliore dei requisiti.
  - Il prototipo sperimenta le parti del sistema che **non sono** ancora ben comprese
- Realizzazione di una prima implementazione (prototipo), più o meno incompleta da considerare come una 'prova', con lo scopo di:
  - accertare la fattibilità del prodotto
  - validare i requisiti

Modelli basati su prototipo throw-away (1)

- Il prototipo è un mezzo attraverso il quale si interagisce con il committente per:
  - accertarsi di aver ben compreso le sue richieste
  - specificare meglio tali richieste
  - valutare la fattibilità del prodotto
- Dopo la fase di utilizzo del prototipo, questo viene buttato (throw-away) e si passa alla produzione della versione definitiva del Sistema SW mediante un modello che, in generale, è di tipo a cascata

Modelli basati su prototipo throw-away (2)



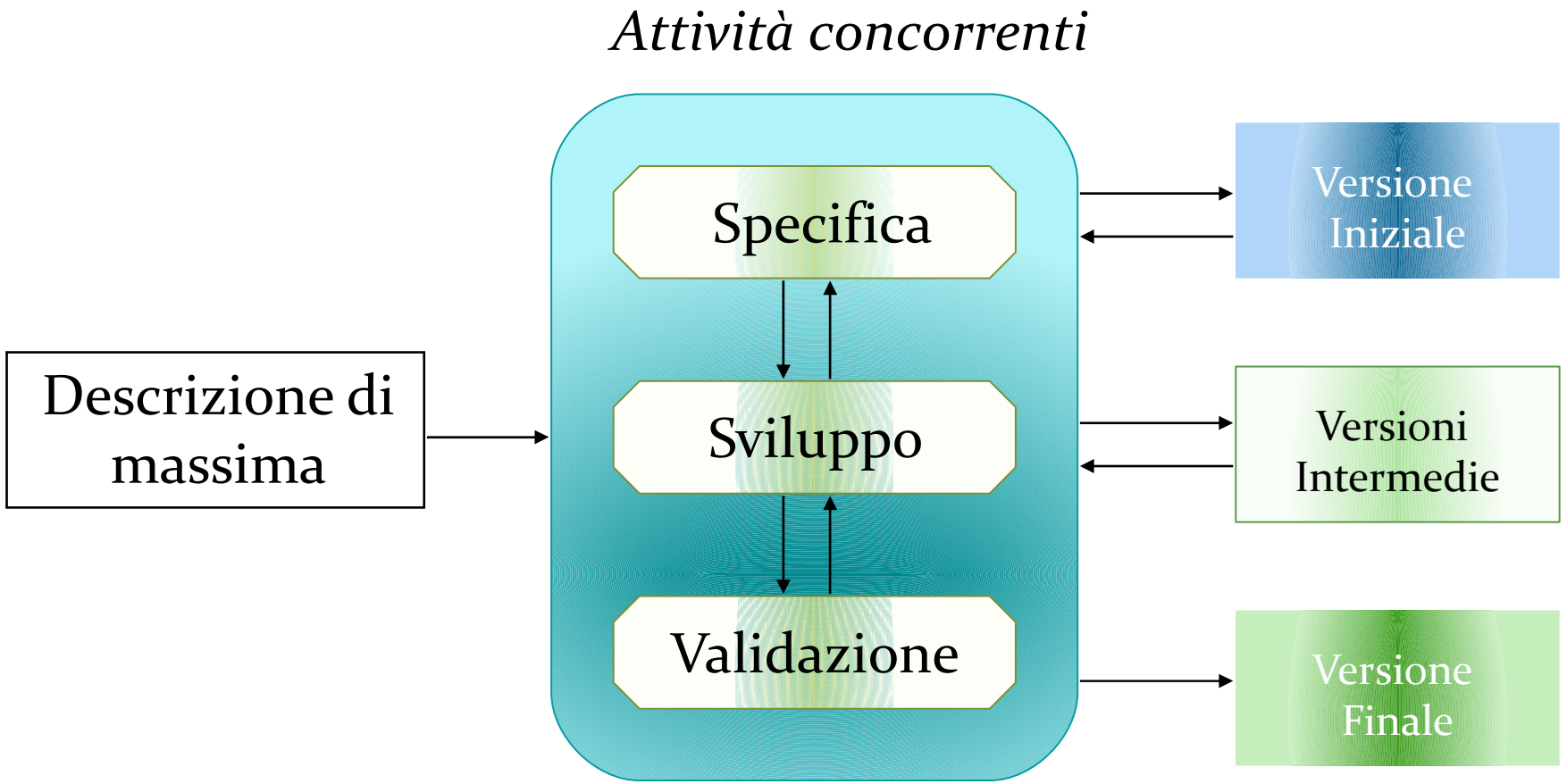
- Prototipi usa e getta
  - *il prototipo è uno strumento di per validare i requisiti e/o validarne la fattibilità;*
    è incompleto, approssimativo, realizzato utilizzando parti già possedute
  - *Il prototipo **deve** essere gettato !*

Tipi di prototipi throw-away

- Esistono due tipi di prototipi usa&getta:
  - **Mock-ups**: produzione completa dell'interfaccia utente. Consente di disambiguare parte dei requisiti, grazie all'interazione col cliente.
    - Si può, già in questa fase, definire il manuale di utente
  - **Breadboards**: implementazione di sottoinsiemi di funzionalità critiche del software, per validarne vincoli e fattibilità (carichi elevati, tempo di risposta, ...), senza le interfacce utente.
    - Produce feedbacks su come implementare la funzionalità (in pratica si cerca di conoscere prima di garantire).

I Modelli Evolutivi

Modelli evolutivi

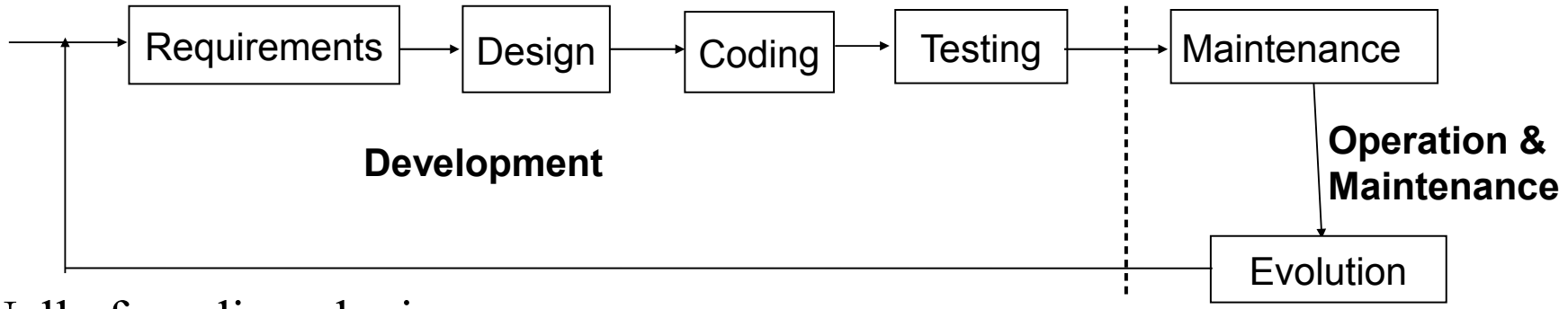


Modello evolutivo

- Prototipazione di tipo evolutivo (sviluppo esplorativo)
  - L'obiettivo è lavorare con il cliente ed evolvere verso il sistema finale a partire da una specifica di massima.
  - Lo sviluppo inizia con le parti del sistema che **sono** già ben specificate, aggiungendo via via nuove caratteristiche

Modello evolutivo: Sviluppo esplorativo

- Inizio a sviluppare ciò che mi è ben chiaro del problema.



Nella fase di evoluzione:

- si analizza l'esperienza di uso del SW sul campo e si utilizza la maggiore conoscenza per definire nuovi obiettivi,
- si determinano esigenze e nuove funzionalità emerse o non coperti,
- si riprende il ciclo dalla definizione dei requisiti alla messa in esercizio e manutenzione del nuovo SW nato dall'arricchimento ed evoluzione del precedente.

Modello evolutivo: Sviluppo esplorativo

- Problemi
  - Mancanza di visibilità del processo
  - Sistemi spesso poco strutturati
  - Possono essere richieste particolari capacità (ad esempio in linguaggi per prototyping rapido)

- Applicabilità
  - Sistemi interattivi di piccola o media dimensione
  - Per parti di sistemi più grandi (es. interfaccia utente)
  - Per sistemi a vita breve

I modelli incrementali

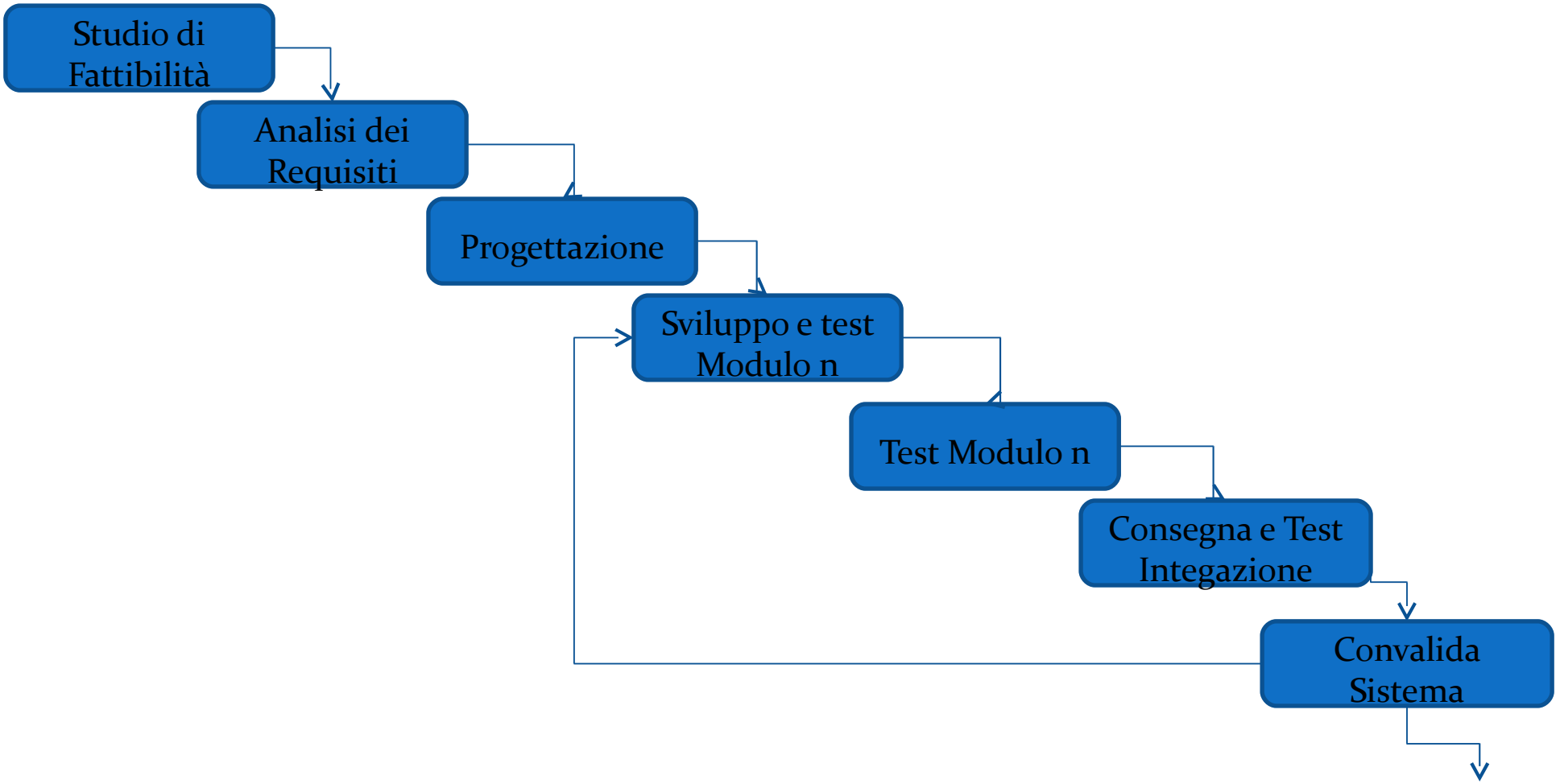
Modelli Incrementali

- Risolvono la difficoltà a produrre l'intero Sistema in un blocco unico nel caso di grandi progetti SW (sia per problemi del produttore che del committente - quest'ultimo potrebbe non avere l'immediata disponibilità finanziaria necessaria per l'intero progetto)
- Un approccio sistematico alla costruzione per parti: Modelli incrementali
  - modello a consegna incrementale
  - modello a spirale

Modello a consegna incrementale

- Le fasi alte del Waterfall Modell sono completamente realizzate.
  - Il SW viene totalmente definito nei requisiti, specificato e progettato.
- I sottosistemi vengono implementati, testati, rilasciati, installati e messi in manutenzione secondo un piano di priorità in tempi diversi.
- Diventa fondamentale la fase di integrazione di nuovi sottosistemi prodotti con quelli già in esercizio

Modello a consegna incrementale

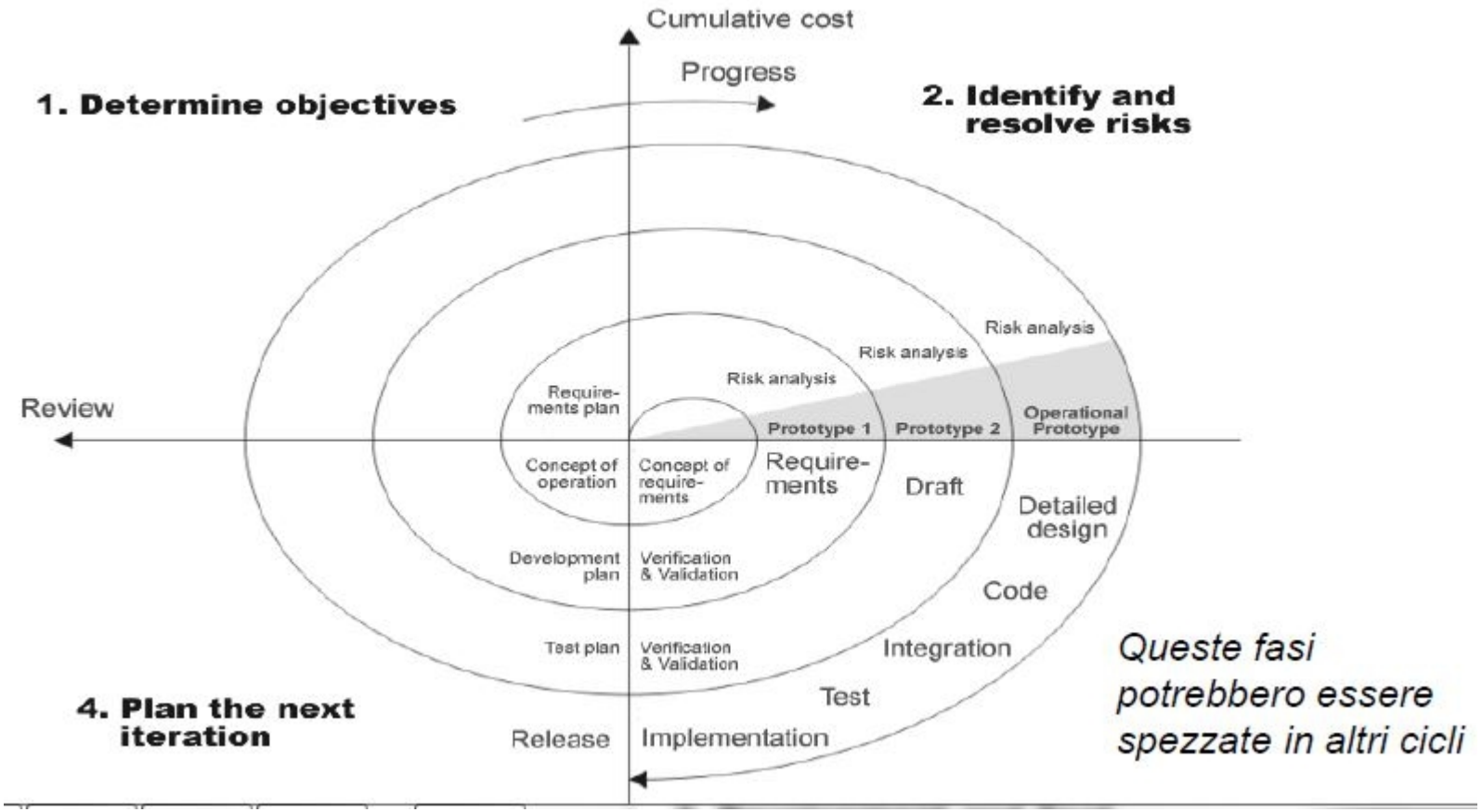


Modello a spirale

- **Formalizzazione del concetto di modello incrementale**
- Ripetizione ciclica di task region:
  - Determinazione obiettivi, vincoli, alternative
  - Valutazione alternative, analisi dei rischi
  - Sviluppo, verifica e convalida
  - Pianificazione prossimo ciclo
- Possibilità di utilizzare uno o più modelli
  - Il ciclo a cascata si può vedere come un caso particolare (una sola iterazione)

Modello a spirale di Boehm

Modello a spirale



Rischi

- Fattori variabili che influiscono negativamente sulla riuscita di un progetto
  - Di tipo tecnico e non tecnico
- Esempi:
  - personale non qualificato per le attività da effettuare
  - budget e pianificazione irrealistici
  - errori di definizione delle funzionalità
  - errori di definizione dell'interfaccia utente
  - funzionalità non richieste ma costose (gold planting)
  - instabilità dei requisiti
  - difetti in componenti sviluppati da terze parti
  - errori in attività svolte all'esterno dell'azienda
  - uso di tecnologie non consolidate o inaffidabili

Gestione dei rischi

- Compito di chi gestisce (il manager) è minimizzare i rischi
- Il rischio è insito in tutte le attività umane ed è una misura dell'incertezza sul risultato dell'attività
- Alti rischi provocano ritardi e costi imprevisti
- Il rischio è collegato alla quantità e qualità delle informazioni disponibili: meno informazione si ha più alti sono i rischi

I modelli e la gestione dei rischi

- **Cascata**
  - alti rischi per sistemi nuovi, non familiari per problemi di specifica e progetto
  - Bassi rischi nello sviluppo di applicazioni familiari con tecnologie note
- **Prototipazione**
  - bassi rischi per le nuove applicazioni, specifica e sviluppo vanno di pari passo
  - alti rischi per la mancanza di un processo definito e visibile
-

Esempio

- Obiettivi
  - Migliorare la qualità del software in modo significativo
- Vincoli:
  1. In tre anni
  2. Senza grandi investimenti
  3. Senza un cambiamento radicale degli standard dell'azienda
- Alternative
  1. Riutilizzare software certificato già esistente
  2. Introdurre specifiche e verifiche formali
  3. Investire in prodotti di testing e di validazione

Esempio (cont.)

- Rischi
  - Migliorare la qualità del software può aumentare eccessivamente i costi
  - I nuovi metodi possono indurre problemi di personale
- Risoluzione dei rischi
  1. Studio della letteratura esistente
  2. Avvio di un progetto pilota
  3. Analisi delle componenti potenzialmente riutilizzabili
  4. Valutazione degli strumenti di supporto già esistenti
  5. Addestramento e motivazione del personale

Esempio (cont)

- Risultati

1. I miglioramenti sono difficili da quantificare per la scarsa esperienza nell'utilizzo di metodi formali
2. Gli strumenti di supporto disponibili sono insufficienti rispetto allo standard dei sistemi di sviluppo dell'azienda
3. Componenti software riusabili, ma scarso contributo degli strumenti di supporto alla riusabilità

- Pianificazione della fase successiva
  - Finanziare una fase di studio di altri 18 mesi
  - Studiare in maggior dettaglio le opzioni di riuso del software
    Sviluppare strumenti di supporto al riuso di software
    Esplorare uno schema di certificazione delle componenti

Modello a spirale: vantaggi e svantaggi

- **Vantaggi**
  - Rende esplicita la gestione dei rischi
  - Focalizza l'attenzione sul riuso
  - Aiuta a determinare errori nelle fasi iniziali
  - Obbliga a considerare gli aspetti di qualità
  - Integra sviluppo e manutenzione
  - Costituisce un framework di sviluppo hardware/software

eXtreme Programming

- Metodologia adeguata a piccolo ti team di sviluppo che devono sviluppare il SW in modo veloce in un ambiente in evoluzione dinamica.
- Tende a introdurre e rendere espliciti gli elementi do progettazione nel codice stesso al fine di ridurre il più possibile la documentazione
- La progettazione del sistema ha carattere incrementale
- Non si ha una progettazione complessiva dell'architettura predisposta ad ospitare il complesso dei requisiti
  - I requisiti sono trattati uno alla volta incrementalmente
  - Preliminare è il refactoring per ospitare il requisito corrente
  - Il sistema risultante dal refactoring è sottoposto a test
  - Vengono preparati nuovi test per il requisito
  - Viene implementato il requisito

eXtreme Programming: Principi chiave

- **Rapid Feedback**: affrontare i problemi prima permete dia ver più tempo per risolverli (feedback da clienti ed attività di testing).
- **Simplicity**: La progettazione dovrebbe concentrarsi solo sui requisiti di interesse corrente (altri requisiti gestiti quando occorreranno). Nessuna predisposizione anticipata al cambiamento.
- I**ncremental Change**. I cambiamenti vanno affrontati uno alla volta (piuttosto che simultaneamente) integrandoli al sistema corrente.
- **Embracing chang**e: Il cambiamento è un evento frequente ed usuale in XP; non va trattato come un fenomeno eccezionale.
- **Quality work**. XP si focalizza su progetti rapidi dove gli avanzamenti sono mostrati frequentemente. Cura massima va comunque dedicata alla stesura di ogni avanzamento (attenzione agli aspetti qualitativi).

eXtreme Programming: Pianificazione (1)

- **Requisiti**: I requisiti sono raccolti in formati di casi d'uso di alto livello (**Storie**) che raccolgono caratteristiche coerenti.
- **Task di sviluppo**: Le storie sono decomposte in task di sviluppo. La durata dei task è espressa in giorni e non eccede le due settimane.
  - Permette di quantificare la durata della produzione
  - Unità di misura la ideal week
  - Di solito calibrata con fattore correttivo per meeting, vacanze, malattie.
  - **Project velocity**: quante ideal weak sono fatte in una unità di tempo.
-

eXtreme Programming: Pianificazione (2)

- **Release**: Le storie sono assegnate alle varie release del prododotto dopo una operazione di prioritizzazione (fatta dall'utente) delle funzionalità (**release plan**)
  - Le release sono schedulate frequentemente (1-2 mesi) per avere feedback rapido.
  - Ogni release è costituita da un certo numero di iterazioni del processo
  - Ogni iterazione è sottoposta al committente per validazione
  - Il committente ha opportunità di introdurre molte volte cambiamenti nella fase di sviluppo
  - Per ogni release è definito un acceptance test (dal committente) che viene utilizzato come test di regressione in tutte le release successive.
  - Se i tempi delle iterazioni deviano da quelli previsti (per due iterazioni) si ridefinisce il release plan.

eXtreme Programming: Riuso e specifica

- **XP non enfatizza l'aspetto del riuso come metodo chiave per aumentare la produttività**
  - L'uso dei design pattern non è escluso ma il consiglio è di usare le soluzioni più semplici
  - I design pattern non sono usati preventivamente ma come effetto eventualmente del refactoring.
  - Non è enfatizzata la scelta architetturale, l'architettura emerge dallo sviluppo
- **XP minimizza la produzione di documentazione**
  - La continua interazione col committente sostituisce la specifica dei requisiti
  - Uso di codifiche standard per includere nel codice informazione sulla specifica della struttura
  - La forma di comunicazione degli sviluppatori è il codice

eXtreme Programming: Processo

- **Il ciclo del processo consiste di 4 attività**
  - **Planning** (si colloca all'inizio di ogni iterazione)
  - Le tre attività rimanenti si iterano in rapida successione nella stessa iterazione.
  - **Design**
  - **Coding**
  - **Testing**
- **Vincoli sulle attività:**
- **Test First:** Gli Unit test sono scritti dagli sviluppatori e sono scritti prima della stesura del codice. (costringe a fissare le interfacce delle classi).
- **Write test for each new bug**: Quando si individua un difetto si crea si introduce un test per quel difetto.

eXtreme Programming: Processo (2)

Vincoli sulle attività:

- **Refactor before extending**: Refactoring prima di provvedere ad uan estensione in modo da ospitare al meglio l'estensione. Serve a mantenere la qualità interna.
- **Production code is written in pairs**: Il codice che serve ad una release è sempre sviluppato in coppia in modo da incrementare la qualità del codice
  - La pratica code in pairs assolve la necessità di code inspection e peer review.
  - Migliorare la progettazione e l'assenza di difetti
  - Ccondividera la conoscenza della progettazione
- **Integrate often**: Ilcodice degli sviluppatori è integrato frequentemente nel corpo principale.
  - Gli Unit test sono usati per test di regressione.

eXtreme Programming: Organizzazione

- **Principio:** Riduzione del numero degli incontri per dedicare più tempo alle altre attività.
- **Stand-up meeting giornaliero**: Meeting in piedi in cui si espone il lavoro svolto il giorno precedente
  - In piedi per velocizzare
  - Solo comunicazione e non discussione
- Le coppie di sviluppatorri non sono stabili ma ruotano di progetto in progetto.
- **Non c'e' un manager ma un leader di gruppo** (sistema self-organizing)
  - Il leader incoraggia un ambiente i cui i partecipanti collaborano condividono informazione ed hanno mutua fiducia.
  - Il leader trasmette la visione dei requisiti e dell'architettura e dell'ambiente complessivo di sviluppo

Test Driven Development (TDD)

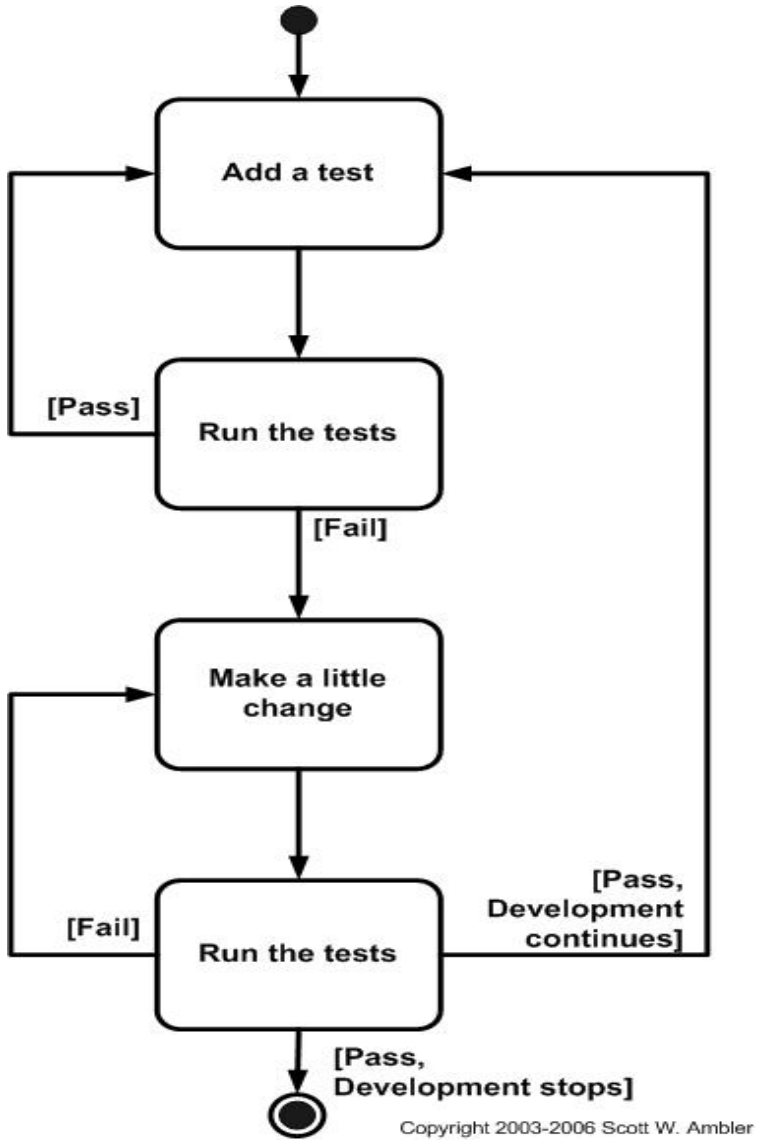
- E' un processo di tipo evolutivo che combina due metodologie di sviluppo
  - **Test-First Development**

    (Si scrive il test prima di scrivere sufficiente codice per eseguirlo)
  - **Refactoring**
- **Due diverse prospettive di osservazione del metodo**
- **Una tecnica di specifica:**
  - Un modo di pensare e formulare i requisiti e gli aspetti di progettazione prima di scrivere il codice.
  - Tecnica **agile requirement e agile design**
- **Una metodologia di programamzione**
  - Un metodo per scrivere 'clean code that works'.

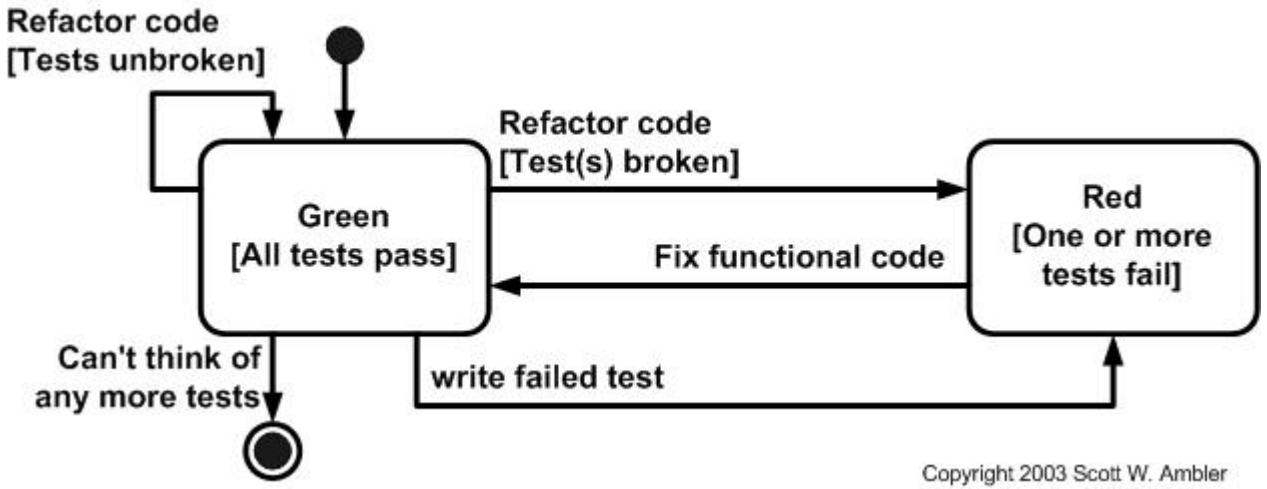
Test_First Development (TFD)

- **1) Aggiungere un caso di test per il quale il codice prodotto fallisca**
- **2) Eseguire l'intero test (o sua parte significativa) per assicyare che il codice non passi il test.**
- **3) Aggiornare il codice per far garantire che il test sia superato**
- **4) Eseguire interamente il test ed eventualmente variare il codice finché il test non é superato.**
- **5) Passare all'iterazione successiva introducendo eventualmente una fase di refactoring**

Test_First Development (TFD)



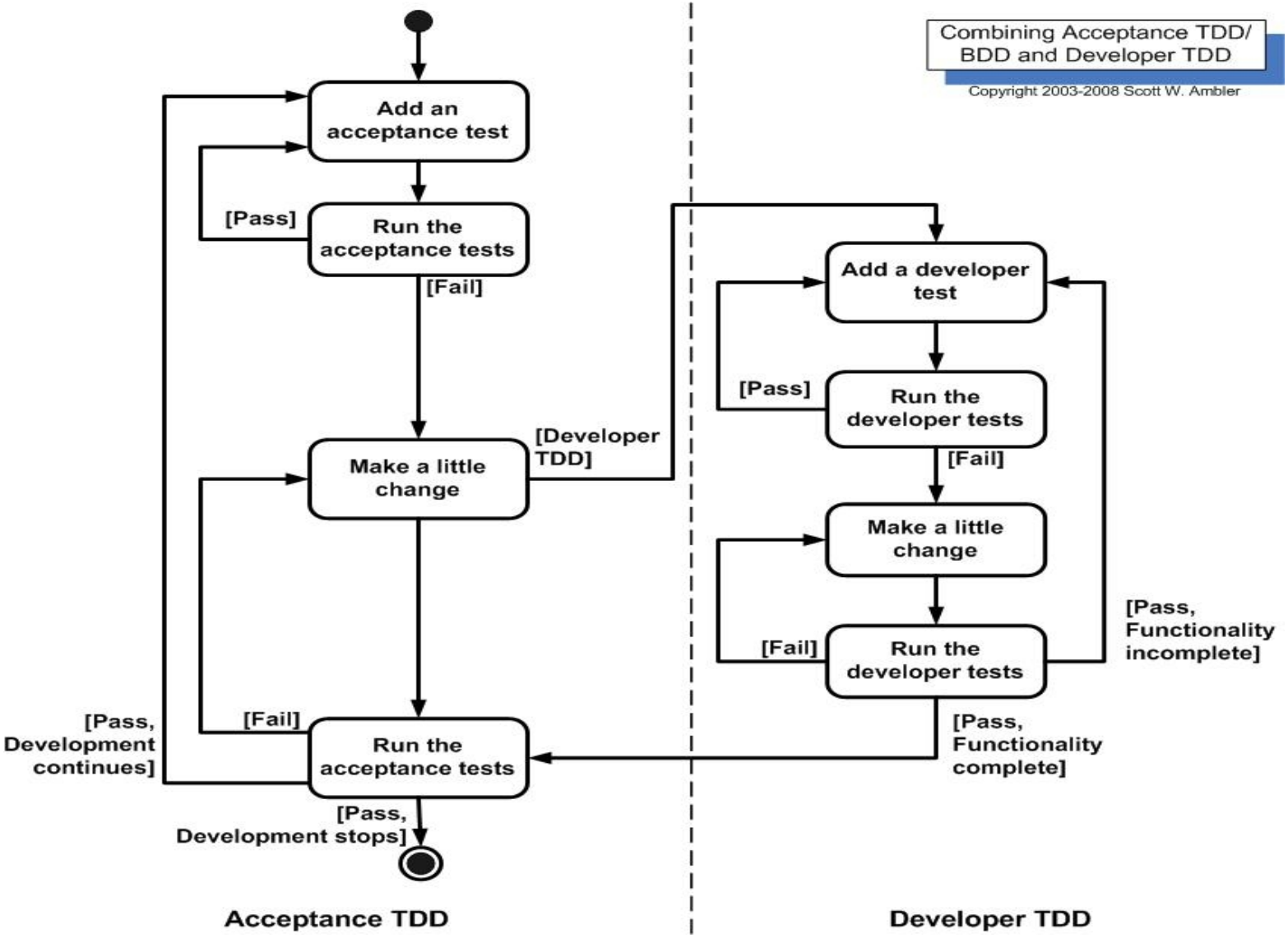
TDD = TFD + Refactoring



Livelli di TDD

- **Acceptance TDD (ATDD)**
  - Si aggiunge un acceptance test (specifica behavioural) e codice e funzionalità sufficienti a farlo passare
  - (incremento di specifica di alto livello)
- **Developer TDD**
  - Si aggiunge un sigolo caso di test di unità eil codice necessario a farlo passare
  - (incremento di specifica di basso livello)
- I due livelli di TDD possono cooperare come mostrato nel diagramma della prossima slide.

ATDD + TDD


Combining Acceptance TDD/ BDD and Developer TDD
Copyright 2003-2008 Scott W. Ambler
Add an acceptance test
[Pass]
Run the acceptance tests
[Fail]
Make a little change
[Developer TDD]
[Pass, Development continues]
[Fail]
Run the acceptance tests
[Pass, Development stops]
Acceptance TDD
Add a developer test
[Pass]
Run the developer tests
[Fail]
Make a little change
[Fail]
Run the developer tests
[Pass, Functionality incomplete]
[Pass, Functionality complete]
Developer TDD

Prerequisiti

- **Prerequisiti sulle competenze**
  - I tecnici impiegati nel processo di sviluppo che adotta TDD e/o ATDD devono avere capacità e competenze specifiche nel campo del testing.

- **Ambiente di sviluppo**
  - La metodologia non può essere supportata senza un adeguato framework di testing
  - Per ATDD ad esempio Fitnesse o Rspec
  - Per developer TDD la famiglia Xunit di tool open source (ad es Junit).

Modalità di comportamento individuali

- Il codice eseguibile è il feedback tra passi decisionali
- Lo sviluppatore provvede a generare autonomamente i casi di test
- L'ambiente di sviluppo deve essere efficiente per piccoli cambiamenti (compliatori veloci, suite per test di regressione etc)
- La progettazione deve prevedere componenti fortemente coese e scarsamente accoppiate per agevolare le operazioni di test.

Caratteristiche desiderabili dei casi di test

- Esecuzione veloce (setup, run time e break down brevi)
- Esecuzione in isolamento (si deve poterne alterare l'ordine di esecuzione)
- Usare dati e nomi che renda facile la lettura e la loro comprensione
- Usare dati reali quando possibile nei casi di test
- Ogni caso di test dovrebbe rappresentare un passo significativo di avanzamento

Casi di test come documentazione

- I casi di test non possono sostituire integralmente la documentazione ma possono costituire una parte significativa della documentazione.
- E' una forma di documentazione adeguata agli sviluppatori (gli sviluppatori preferiscono la lettura del codice alla lettura della documentazione scritta)
- In linea con lo spirito di una documentazione agile.
- I test di accettazione definiscono esattamente quello che gli stackholder si aspettano dal sistema e dunque specificano i requisiti critici
- **Acceptance Test come specifiche eseguibili del sistema**
- **Test di Unità come specifiche eseguibili del codice.**
- **I test di Unità possono divenire una porzione significativa della documentazione.**

Limitazioni del TDD

- Pur essendo centrata e guidata dall'attività di testing, l'attività di testing essendo principalmente concentrata sul test di unità, non può esaurire l'attività di test necessaria.
- La metodologia TDD è specialmente legata alla fase di codifica e dunque meno adatta a gestire le fasi di più alto livello della gestione del processo software
- Per poter gestire il processo di gestione complessivo del processo di sviluppo software la metodologia può essere abbinata ad una metodologia agile di sviluppo (Agile Model Driven Development).

TDD e Agile Model Driven Development

- Identify the high-level scope
- Identify initial "requirements stack"
- Identify an architectural vision

- Modeling is part of iteration planning effort
- Need to model enough to give good estimates
- Need to plan the work for the iteration

- Work through specific issues on a JIT manner
- Stakeholders actively participate
- Requirements evolve throughout project
- Model just enough for now, you can always come back later

- Develop working software via a test-first approach
- Details captured in the form of executable specifications