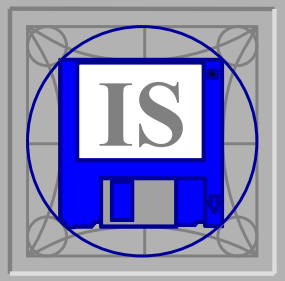
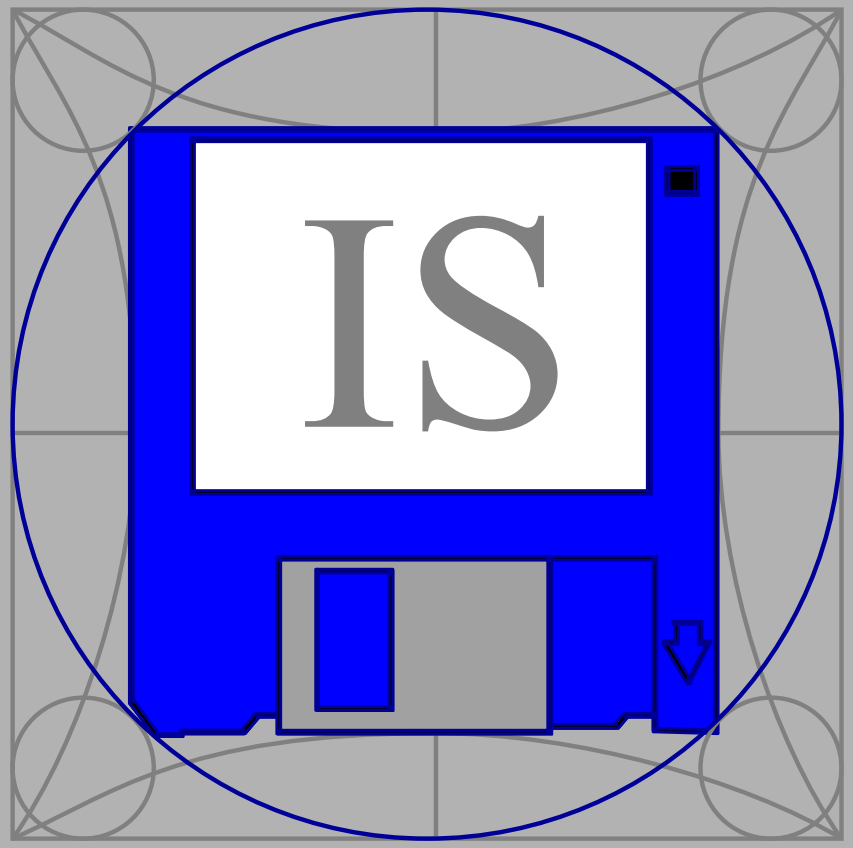


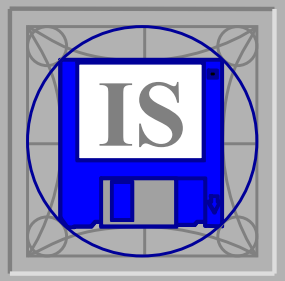
Il ciclo di vita del SW



Ingegneria del Software

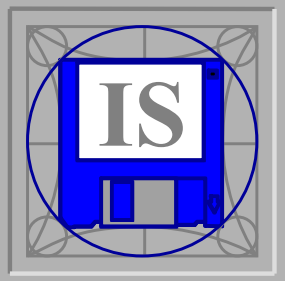
V. Ambriola, G.A. Cignoni,
C. Montangero, L. Semini

Aggiornamenti : T. Vardanega (UniPD)



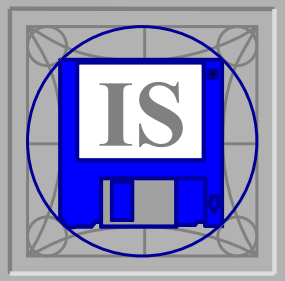
Il concetto di ciclo di vita – 1/2

- Concezione → sviluppo → utilizzo → ritiro
  - Questi sono i principali stati di vita di un prodotto SW
  - A noi qui interessa solo il segmento [concezione → sviluppo]
- La transizione tra stati (arco di automa a stati finiti) avviene per azione di processi di ciclo di vita
  - Quelli di cui abbiamo parlato nella lezione precedente
- Un progetto fa progredire lo stato di avanzamento di un prodotto SW
- Per quel fine, il progetto mobilita specifiche attività di specifici processi
  - Attività che ordiniamo in funzione delle dipendenze tra i loro ingressi e le loro uscite
  - Fissando così i criteri di loro attivazione e di completamento



Il concetto di ciclo di vita – 2/2

- Il termine «fase» corrisponde allo stazionamento in uno stato di ciclo di vita o in una transizione tra stati
  - Essa designa uno stato consistente, entro un dato segmento temporale
  - Per questo, «fase» non è sinonimo di «attività»
- Esistono molteplici cicli di vita, che differiscono tra loro per transizioni tra stati e regole di attivazione
  - Ciascuno di essi viene idealizzato da un «modello»
- Aderire a un particolare modello comporta vincoli sulla pianificazione e gestione del corrispondente progetto
  - Questo influenza la selezione del *way of working* e dei suoi strumenti di supporto
- In questo corso, quindi, ci occupiamo di modelli di sviluppo SW

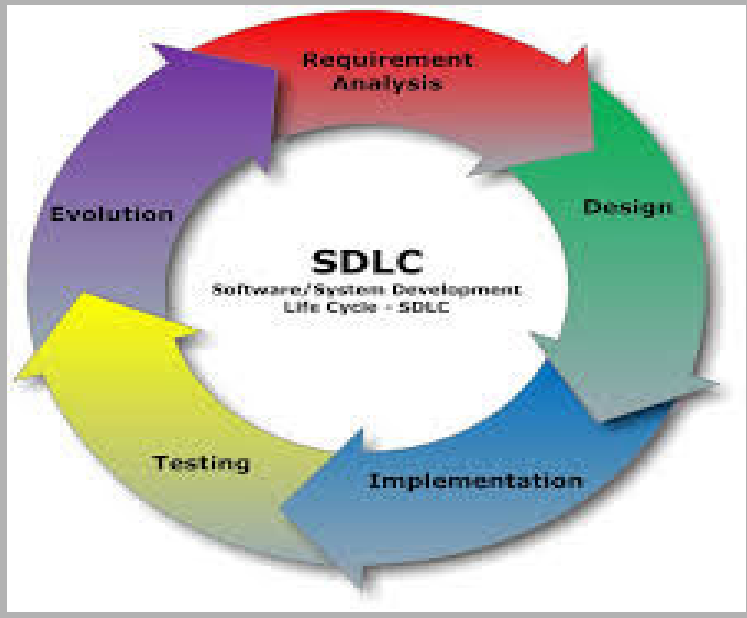


Cosa significa "modello"

- Insieme di specifiche che descrivono un fenomeno di interesse (astratto o concreto)
  - In modo che non dipende dall'osservatore
  - Dimostrato corretto (empiricamente o per teorema)
- I modelli aiutano a studiare, comprendere, misurare, trasformare l'oggetto di interesse
  - Il modello specifica cosa esso sia
  - L'architettura (*design*) specifica come esso funzioni
  - L'analisi (del modello) spiega perché quell'oggetto fa quel che fa nel modo in cui lo fa

Esempio

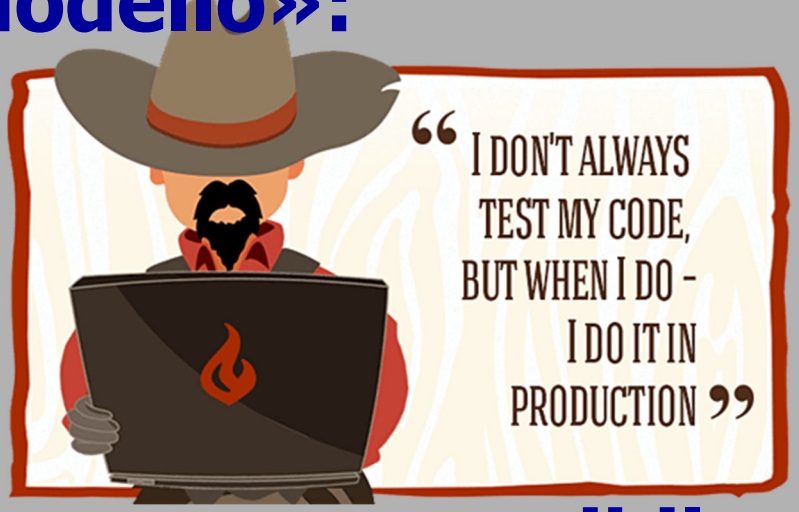
- Questo è un modello di ciclo di vita che non contempla «ritiro» (fine vita)



- Lo stadio di «evoluzione» (manutenzione evolutiva) innesca nuovi attraversamenti di ciclo

Modelli di sviluppo – 1/2

- **Alle origini vi fu il «non-modello»: *Code-'n-Fix***
  - **Aka " *Cowboy coding* "**



- **Attività senza organizzazione comprensibile dall'esterno**
  - **L'esatto contrario di sistematico, disciplinato, quantificabile**

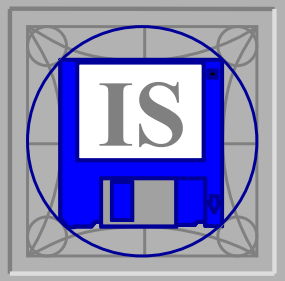
Modelli di sviluppo – 2/2

- Quello stile causò la crisi del SW, che portò alla nascita della disciplina SWE
- Ne nacque una successione di modelli organizzati

Modello	Tratti caratteristici
Cascata	Rigide fasi sequenziali
Incrementale	realizzazione in più passi
A componenti	Orientato al riuso
Agile	Altamente dinamico, fatto di brevi cicli iterativi e incrementali

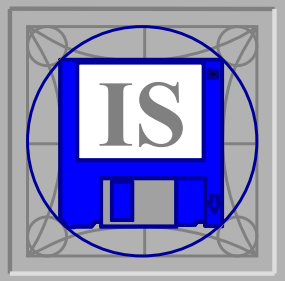
Glossario

- **Iterazione**
  - Raffinamenti o rivisitazione (pittura): distruttivo
- **Incremento**
  - Aggiunte successive a un impianto base (scultura): costruttivo
- **Prototipo**
  - Provare e scegliere soluzioni: usa-e-getta o per incrementi
- **Riuso**
  - Copia-incolla opportunistico (occasionale: basso costo, scarso impatto)
  - Sistematico (per progetto / famiglia di prodotti / ogni prodotto): maggior costo, maggior impatto



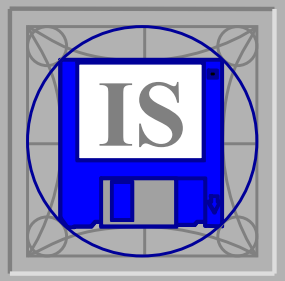
Modello sequenziale (a cascata) – 1/3

- **Definito nel 1970 da Winston W. Royce**
  - "*Managing the development of large software systems: concepts and techniques*"
  - Centrato sull'idea di processi ripetibili
- **Successione di fasi rigidamente sequenziali**
  - Non ammette ritorno a fasi precedenti: eventi eccezionali riportano all'inizio
  - Le iterazioni costano troppo: non sono viste come buon mezzo di mitigazione delle incertezze di sviluppo
- **Prodotti**
  - Principalmente documenti, fino poi a includere il SW



Modello sequenziale (a cascata) – 2/3

- L'ingresso in uno stato di avanzamento è vincolato da pre-condizioni (*gate*)
  - Che devono essere soddisfatte – in modo dimostrabile – dalle post-condizioni delle transizioni che portano in esso
- Il progetto è una successione di fasi distinte, non sovrapposte nel tempo
- Usato per lo sviluppo di sistemi complessi, soprattutto sul piano organizzativo



Modello sequenziale (a cascata) – 3/3

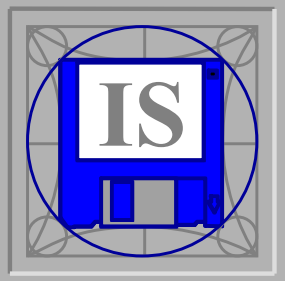
- Ogni fase (stato/transizione) viene definita da
  - Attività previste e prodotti attesi in ingresso e in uscita
  - Contenuti e struttura di documenti che descrivono lo stato raggiunto e le attività svolte
  - Responsabilità e ruoli coinvolti nelle attività
  - Scadenze di consegna dei prodotti
- Entrare, uscire, stazionare in una fase comporta lo svolgimento di determinate azioni
  - Realizzate come attività di specifici processi

Schema secondo ISO 12207:1995

Alle attività qui elencate se ne aggiungono molte altre istanziate da altri processi (documentazione, verifica, configurazione, gestione di progetto, ...)

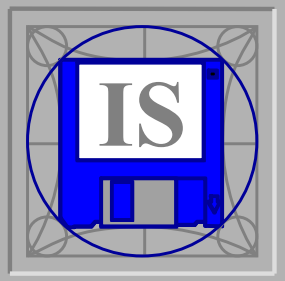
Legenda
AR: analisi dei requisiti
DA: progetto (*design*) architetturale
DD: progetto (*design*) di dettaglio

La manutenzione comporta ritorni sulle fasi precedenti



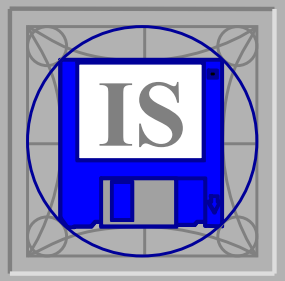
Critica del modello sequenziale

- **Difetto principale: eccessiva rigidità**
  - **Stretta sequenzialità: nessun parallelismo e nessun ritorno**
  - **Non ammette modifiche nei requisiti in corso d'opera**
  - **Visione rigida (burocratica) e poco realistica del progetto**
- **Correttivo 1: con prototipazione**
  - **Prototipi di tipo "usa e getta"**
    - Per capire meglio i requisiti o le soluzioni, strettamente all'interno di singole fasi
- **Correttivo 2: con ritorni**
  - **Come «allenamenti» prima dell'atto definitivo**
    - Per imparare a fare sempre meglio ciò che serve a realizzare il prodotto



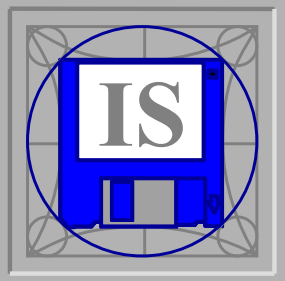
Ritorni: iterazione o incremento?

- Problemi molto complessi richiedono di procedere a tentoni
  - Spesso tramite iterazioni potenzialmente distruttive
- Meglio procedere per piccoli passi incrementali
  - Evitando di integrare il prodotto tutto-in-una-volta (*aka big-bang-integration*)
  - Assai meglio adottare l'integrazione continua
- Iterazione e incremento coincidono quando la sostituzione raffina ma non ha impatto sul resto



Vantaggi dei modelli incrementali

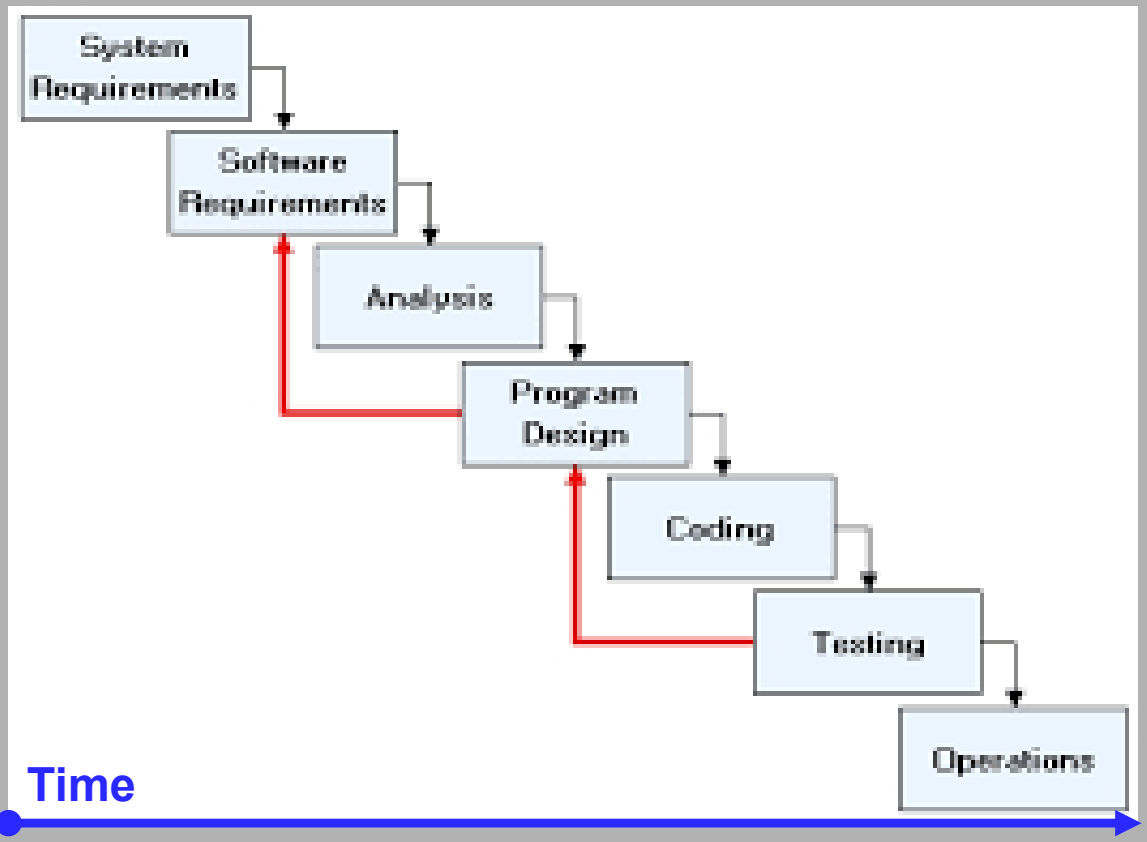
- Possono produrre valore a ogni incremento
  - Un insieme crescente di funzionalità utili diventa presto e progressivamente disponibile
  - Magari a valle di un buona prototipazione, non usa-e-getta
- Procedere per incrementi riduce il rischio di fallimento
  - Senza però azzerarlo …
  - Come un ciclo `for`, da cui sappiamo quando usciremo, a meno di eccezioni
- Le funzionalità fondamentali (più necessarie) vanno sviluppate prima
  - Il loro uso frequente aiuta a verificare che siano solide



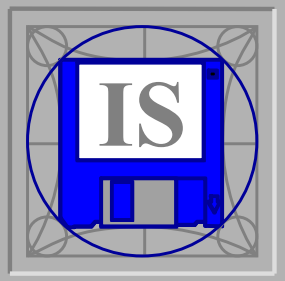
Vantaggi dei modelli iterativi

- Possono applicare a qualunque modello di sviluppo
  - Comportando forte potenziale distruttivo
- Consentono maggior capacità di adattamento
  - Insorgere di problemi, cambio di requisiti, collasso tecnologico
- Comportano il rischio di non convergenza
  - Come un ciclo `while`, da cui non sappiamo per certo se e quando usciremo
- Tecniche di mitigazione
  - Decomporre il sistema in parti, lavorando prima su quelle più critiche, perché più complesse o con requisiti più incerti
  - Fissando un limite superiore al numero di iterazioni

Rischi dei modelli iterativi – 1/2

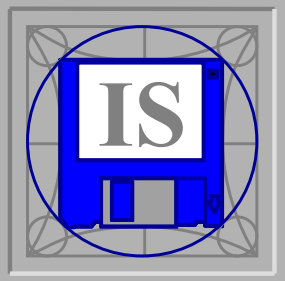


Ogni iterazione comporta un ritorno all'indietro nella direzione opposta all'avanzamento del tempo



Rischi dei modelli iterativi – 2/2

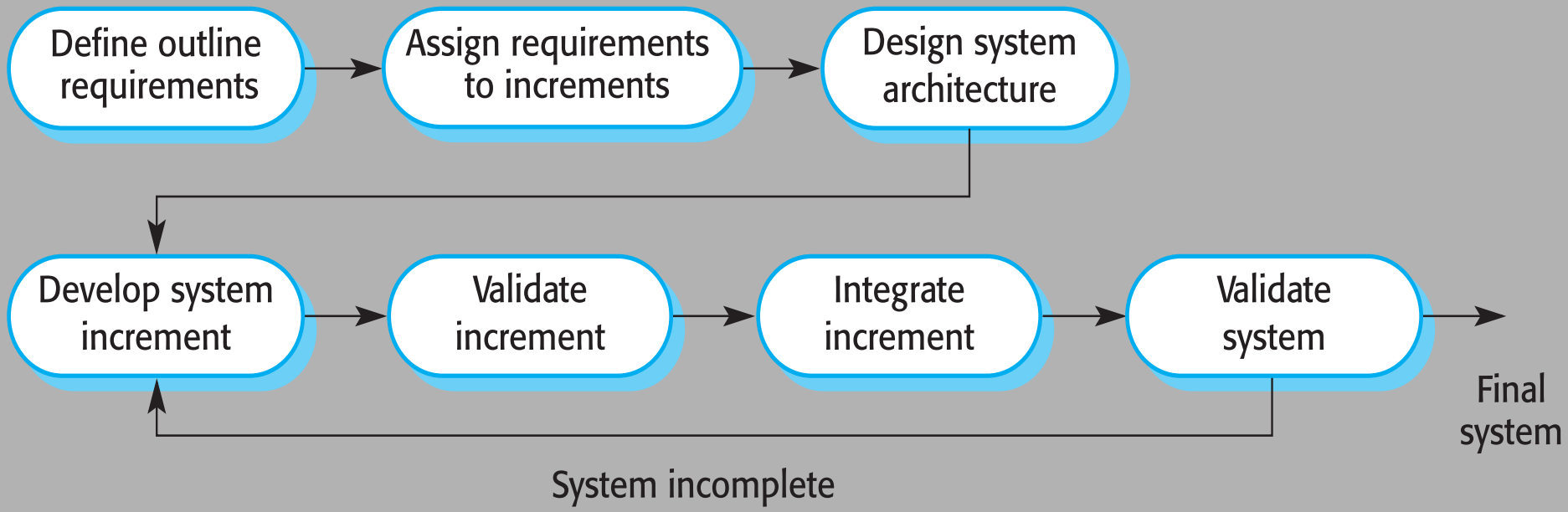
- La nozione di *technical debt* designava in origine parti di sviluppo (*design*, codice) bisognose di *refactoring*, cioè di future «passate iterative»
  - Quelle parti costituivano un debito contratto per avanzare più velocemente, ma da saldare al più presto, per non pagarlo, dopo, con interessi composti …
- Oggi essa designa piuttosto tutti i punti dello sviluppo nei quali la soluzione realizzata non concorda con la nostra comprensione corrente di come invece dovrebbe essere
  - Sono detti *kludge* e sono anch'essi debiti da sanare al più presto
- Vedere la risorsa «Per approfondire» associata a questo argomento



Modello incrementale – 1/2

- Prevede rilasci multipli e successivi
  - Ciascuno realizza un incremento di funzionalità
- I requisiti sono classificati e trattati in base alla loro importanza strategica
  - I primi incrementi puntano a soddisfare i requisiti più importanti sul piano strategico
  - Quello che serve avere prima e quindi di più
- Così i requisiti «importanti» diventano presto chiari e stabili, quindi più facilmente soddisfacibili
  - Quelli meno importanti hanno invece più tempo per stabilizzarsi e armonizzarsi con lo stato del sistema

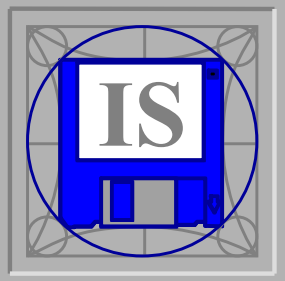
Schema generale



I cicli di incremento sono parte dello sviluppo

La validazione può anch'essa essere incrementale se ogni rilascio è pubblico

Tratto da: Ian Sommerville, *Software Engineering*, 8th ed.



Modello incrementale – 2/2

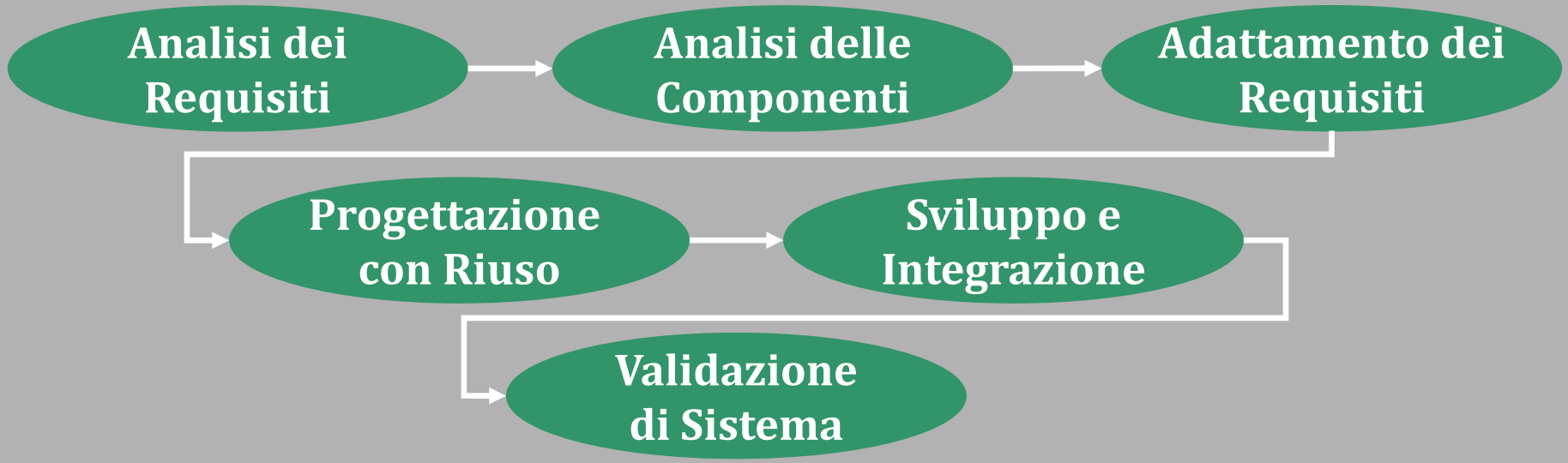
- **Analisi dei requisiti e progettazione architetturale vengono svolte una sola volta**
  - **Per stabilizzare presto i requisiti principali**
  - **Per stabilizzare presto l'architettura complessiva del sistema, che deve essere capace di accogliere incrementi senza rompersi**
  - **Per poter decidere preventivamente il numero di incrementi e i loro specifici obiettivi**

Su questo ritorneremo più avanti

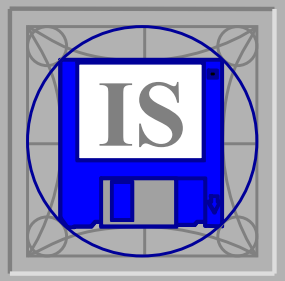
- **La realizzazione è per passi incrementali**
  - **Raffinando l'analisi dei requisiti e la progettazione di dettaglio, strettamente entro l'architettura adottata**
  - **Il completamento dei primi incrementi serve a rendere disponibili le principali funzionalità**

Schema secondo ISO 12207:1995

Modello a componenti

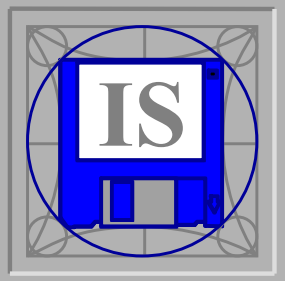


- Molto di quello che ci serve fare è già stato fatto
- Molto di quello che faremo potrebbe servirci ancora
  - Analisi dei requisiti guidata dalla possibilità di riuso di quanto già esista
  - Realizzazione che cerca di favorire riuso futuro



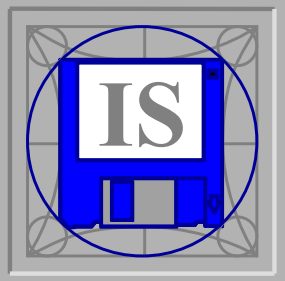
Metodi agili – 1/3

- **Nascono alla fine degli '90 in reazione all'eccessiva rigidità dei modelli allora prevalenti**
  - **http://agilemanifesto.org/**
- **Si basano su quattro principi ideologici**
  - ***Individuals and interactions over processes and tools***
    - L'eccessiva rigidità ostacola l'emergere del valore
  - ***Working sofware over comprehensive documentation***
    - La documentazione non sempre corrisponde a SW funzionante
  - ***Customer collaboration over contract negotiation***
    - L'interazione con gli stakeholder va incentivata e non ingessata
  - ***Responding to change over following a plan***
    - La capacità di adattamento al cambiare delle situazioni è importante



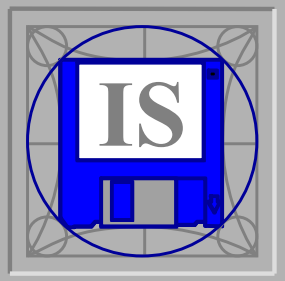
Contro-argomentazioni

- SW privo di documentazione produce costo, non valore
  - Commentare il codice non basta → serve spiegare e motivare le scelte realizzative
- Senza un piano, non si possono valutare rischi e avanzamenti
  - La sola misurazione di consuntivo non può bastare
- Cambiare si può, ma con consapevolezza del rapporto costo/benefici



Metodi agili – 2/3

- L'idea base è il concetto di "*user story*"
  - Una funzionalità significativa che l'utente vuole realizzare con il SW richiesto: uno scenario d'uso
- Ogni "*user story*" è definita da
  - Un documento di descrizione del problema individuato
  - Il verbale delle conversazioni con gli *stakeholder* effettuate per discutere e comprendere il problema
  - La strategia da usare per confermare che il SW realizzato soddisfi gli obiettivi di quel problema

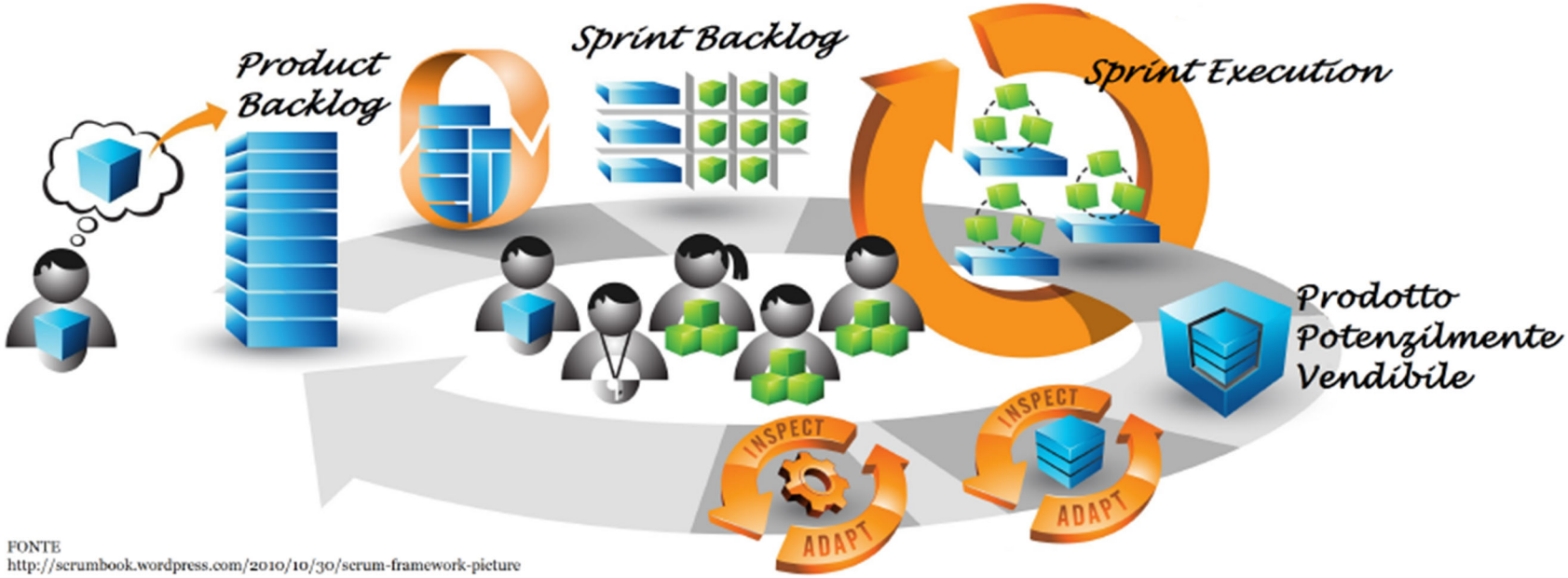


Metodi agili – 3/3

- **Assunti base**
  - Suddividere il lavoro in piccoli incrementi a valore aggiunto, magari anche sviluppabili indipendentemente
  - Sviluppare ciascun incremento in sequenza continua dall'analisi all'integrazione
- **Obiettivi strategici**
  - Poter sempre dimostrare al cliente quanto è stato fatto
  - Verificare l'avanzamento tramite progresso reale
  - Dare agli sviluppatori la soddisfazione del risultato
- **Buoni esempi**
  - Scrum, Kanban (*just-in-time*), Scrumban

Il vero modello incrementale è il paradiso (culmine ideale) dell'agile

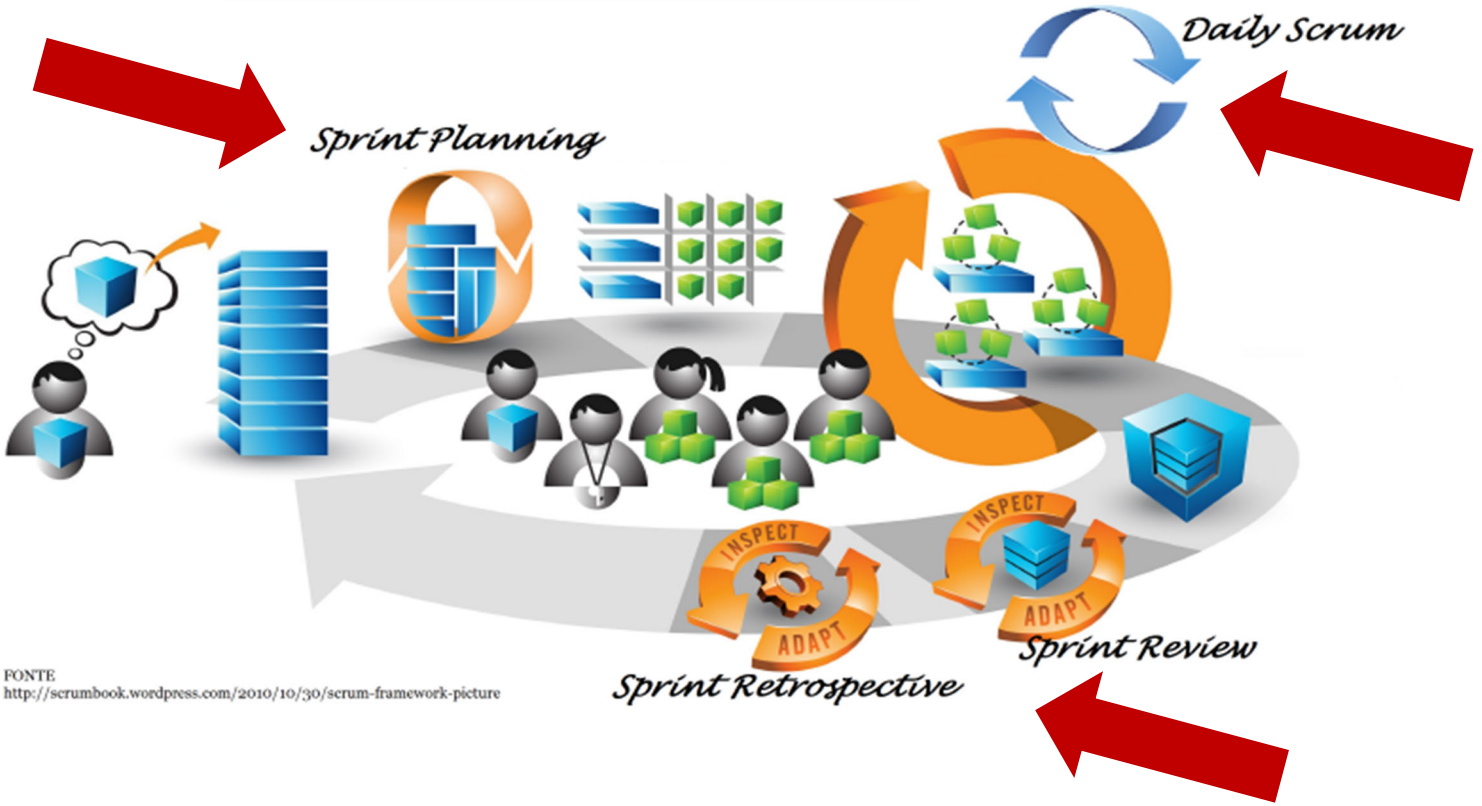
Scrum – 1/2



FONTE
http://scrumbook.wordpress.com/2010/10/30/scrum-framework-picture

- **Product Backlog**
  Requisiti e funzionalità del prodotto
- **Sprint Backlog**
  Insieme di storie del prossimo sprint
- **Sprint**
  Fase operativa di sviluppo
  Durata media 2 - 4 settimane
  Prodotto potenzialmente vendibile

Scrum – 2/2

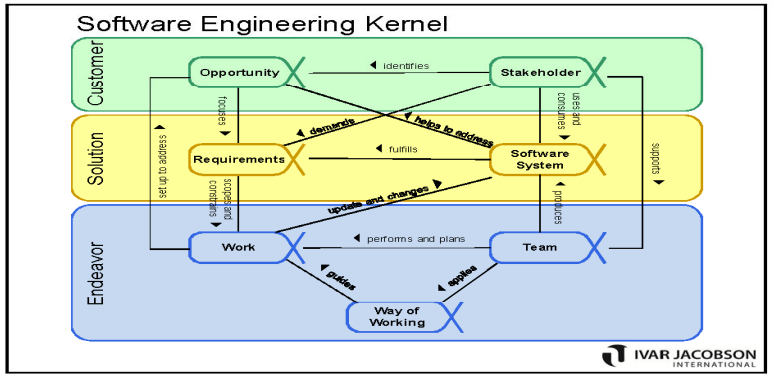


FONTE
http://scrumbook.wordpress.com/2010/10/30/scrum-framework-picture

- **Sprint Planning**
  Pianificazione dello sprint
- **Sprint Review**
  Controllo prodotti dello sprint
- **Daily Scrum**
  Controllo giornaliero avanzamento
- **Sprint Retrospective**
  Controllo qualità sullo sprint

Il ciclo di vita secondo SEMAT – 1/2

IVAR JACOBSON INTERNATIONAL



Stakeholders

- Recognized
- Represented
- Involved
- In Agreement
- Satisfied for Deployment
- Satisfied in Use

The people, groups, or organizations who affect or are affected by a software system.

- Healthy stakeholders represent groups or organizations affected by the software system
- Healthy stakeholder representatives carry out their agreed to responsibilities
- Healthy stakeholder representatives cooperate to reach agreement
- Healthy stakeholders are satisfied with the use of the software system

IVAR JACOBSON INTERNATIONAL

Opportunity

- Identified
- Solution Needed
- Value Established
- Viable
- Addressed
- Benefit Accrued

The set of circumstances that makes it appropriate to develop or change a software system.

- A good opportunity is identified addressing the need for a software-based solution
- A good opportunity has established value
- A good opportunity has a software-based solution that can be produced quickly and cheaply
- A good opportunity creates a tangible benefit

IVAR JACOBSON INTERNATIONAL

Requirements

- Conceived
- Bounded
- Coherent
- Acceptable
- Addressed
- Fulfilled

What the software system must do to address the opportunity and satisfy the stakeholders.

- Good Requirements meets real needs
- Good Requirements has clear scope
- Good Requirements are coherent and well organized
- Good Requirements help drive development

IVAR JACOBSON INTERNATIONAL

www.ivarjacobson.com/semat

Sheet 1 of 2

Il ciclo di vita secondo SEMAT – 1/2

IVAR JACOBSON INTERNATIONAL

Software System

- Architecture Selected
- Demonstrable
- Useable
- Ready
- Operational
- Retired

A system made up of software, hardware, and data that provides its primary value by the execution of the software.

- Good Software System meets requirements
- Good Software System has appropriate architecture
- Good Software System is maintainable, extensible and testable
- Good Software System has low support cost

Team

- Seeded
- Formed
- Collaborating
- Performing
- Adjourned

The group of people actively engaged in the development, maintenance, delivery and support of a specific software system.

- A healthy Team meets its team goals effectively
- A healthy Team has members that collaborates effectively
- A healthy Team focus on their work
- A healthy Team continually improves

Work

- Initiated
- Prepared
- Started
- Under Control
- Concluded
- Closed

Activity involving mental or physical effort done in order to achieve a result.

- Healthy Work is sizeable, estimate-able and track-able
- Healthy Work breakdown reduces dependencies between work items
- Healthy Work management keeps risks, work and re-work under control

Way-of-Working

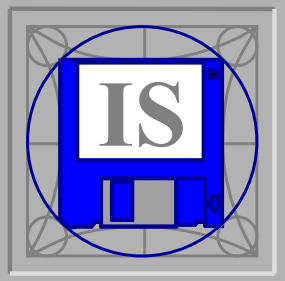
- Principles Established
- Foundation Established
- In Use
- In Place
- Working Well
- Retired

The tailored set of practices and tools used by a team to guide and support their work.

- Good way of working is agreed by the team
- Good way of working reduces risks and technical debts
- Good way of working is effective and removes duplicate work and wastes
- Good way of working improves itself

www.ivarjacobson.com/semat

Sheet 2 of 2



Riferimenti

- W.W. Royce, "*Managing the development of large software systems: concepts and techniques*", Atti della conferenza "Wescon '70", agosto 1970
- B.W. Bohem, "*A spiral model of software development and enhancement*", IEEE Software, maggio 1998
- Center for Software Engineering, http://sunset.usc.edu/research/spiral_model
- ISO/IEC TR 15271:1998, Information Technology – Guide for ISO/IEC 12207
- Scrum: http://www.scrumalliance.org/learn_about_scrum