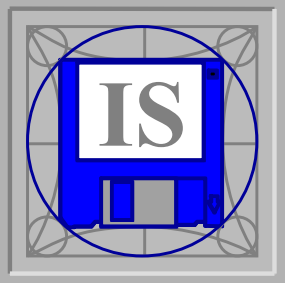
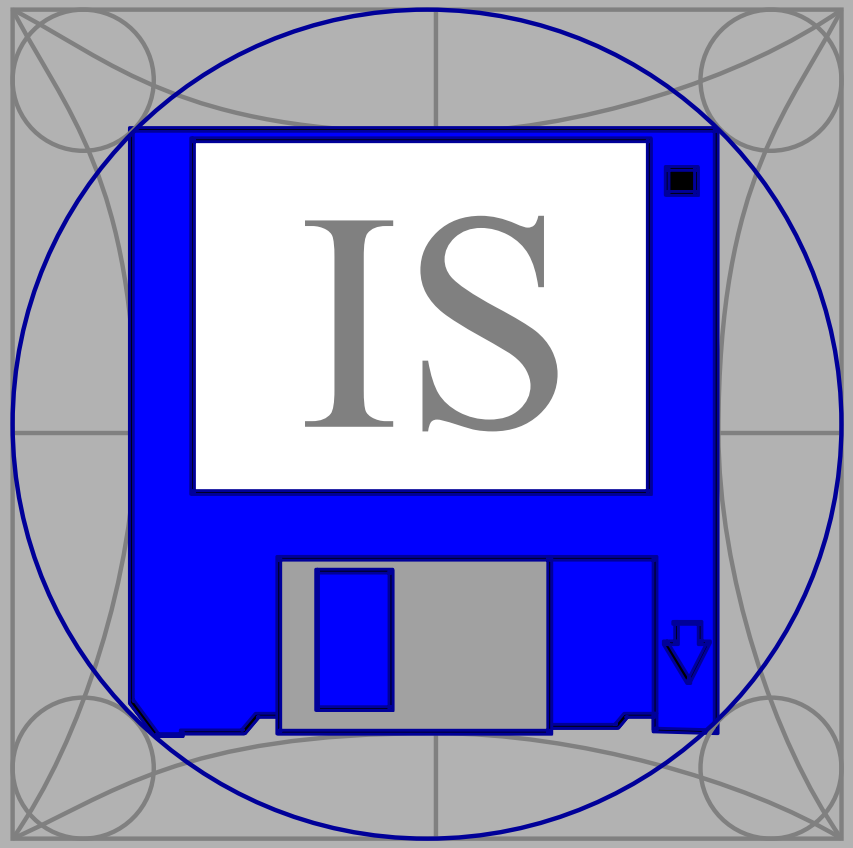


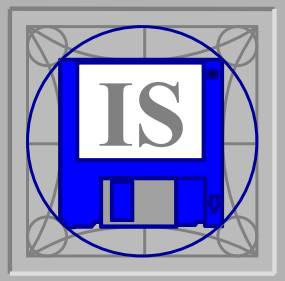
Il ciclo di vita del SW



Ingegneria del Software

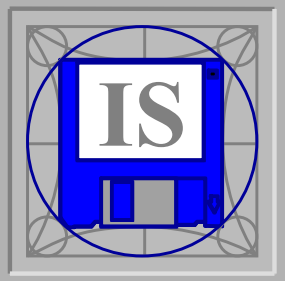
V. Ambriola, G.A. Cignoni,
C. Montangero, L. Semini

Aggiornamenti : T. Vardanega (UniPD)



Il concetto di ciclo di vita – 1

- **Concezione → sviluppo → utilizzo → ritiro**
  - Gli stati (principali) assunti da un prodotto SW durante la sua esistenza
  - Noi qui ci concentriamo sul segmento [concezione → sviluppo]
- **La transizione tra stati avviene tramite l'esecuzione di attività di processi di ciclo di vita**
  - Le attività di processo prevedono specifiche responsabilità, che diventano **ruoli** in un *team* di progetto
- **Per organizzare le attività dei processi attivati nel progetto**
  - Identifichiamo le dipendenze tra i loro ingressi e le loro uscite
  - Fissiamo il loro ordinamento nel tempo e i criteri di attivazione (pre-condizioni) e di completamento (post-condizioni)

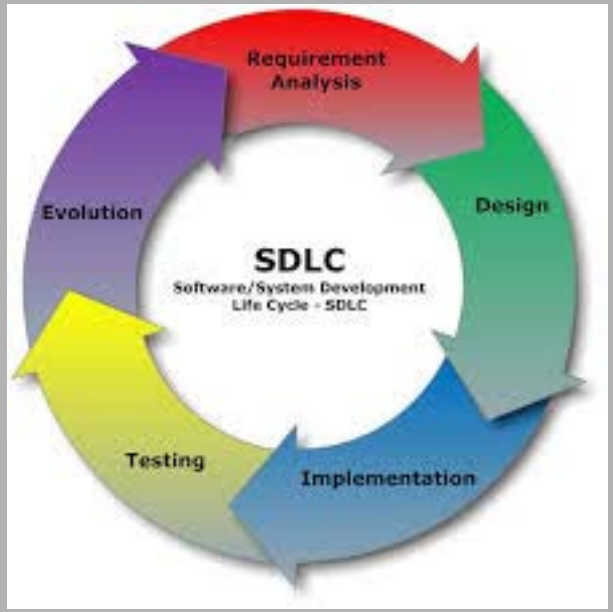


Il concetto di ciclo di vita – 2

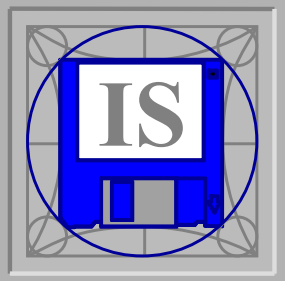
- Lo stazionamento in uno stato di ciclo di vita o in una transizione tra stati viene detta **fase**
  - "Fase" designa un segmento temporale contiguo, con caratteristiche coerenti
- Aderire a un modello di ciclo di vita comporta vincoli sulla pianificazione e gestione del corrispondente progetto
  - La scelta del modello influenza la selezione del *way of working* e dei suoi strumenti di supporto

Modelli di ciclo di vita / di sviluppo

- Parliamo di “modelli” al plurale
  - Esistono molteplici cicli di vita, che differiscono per transizioni tra stati e regole di attivazione
  - L’esempio in figura mostra un ciclo di vita che non prevede ritiro

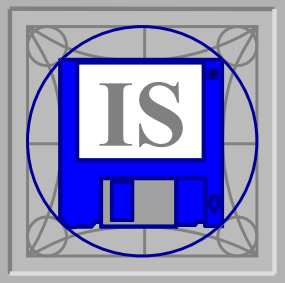


- Come vi sono modelli di ciclo di vita, vi sono modelli di sviluppo
  - Qui ci occupiamo dei secondi
- La manutenzione (quindi la prosecuzione del ciclo di vita del prodotto) istanzierà nuovi mini-cicli di sviluppo



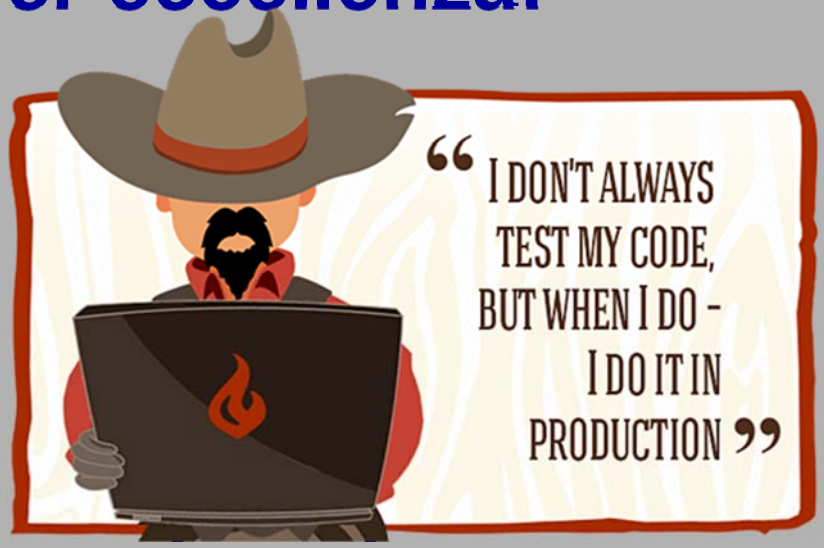
Cosa significa “modello”

- Un insieme di specifiche che descrivono un fenomeno di interesse (astratto / concreto) in modo oggettivo
  - Non dipendente dall’osservatore
  - Dimostrato corretto (empiricamente o per teorema)
- I modelli aiutano ad studiare, comprendere, misurare, trasformare l’oggetto di interesse
  - Il modello specifica cosa esso sia
  - L’architettura interna (*design*) specifica come esso funzioni
  - L’analisi specifica perché fa quel che fa come lo fa

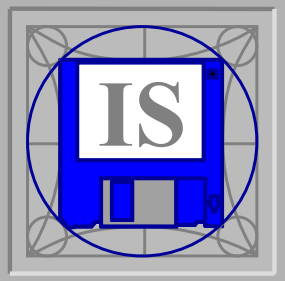


Evoluzione dei modelli di sviluppo – 1

- Si parte dal non-modello per eccellenza: il *Code-'n-Fix* delle origini
  - Aka " *Cowboy coding* "



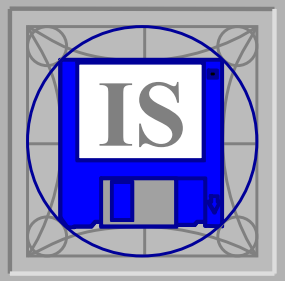
- Insieme di attività senza organizzazione preordinata
  - Fonte di progetti caotici difficilmente gestibili



Evoluzione dei modelli di sviluppo – 2

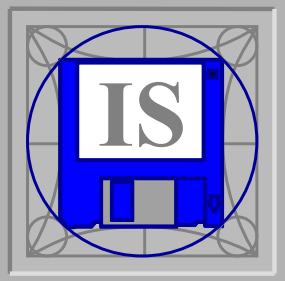
- **Proprio quello stile causa la crisi del SW, che porta alla nascita dell'ingegneria del *software***
- **Nasce una successione di modelli organizzati**

Modello	Tratti caratteristici
Cascata	Rigide fasi sequenziali
Incrementale	realizzazione in più passi
Evolutivo	Ripetute iterazioni che rilasciano diverse varianti di prodotto
A componenti	Orientato al riuso
Agile	altamente dinamico, fatto di brevi cicli iterativi e incrementali



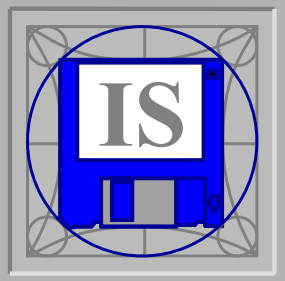
Glossario

- **Iterazione**
  - Procedere per raffinamenti o rivisitazione (pittura): potenzialmente distruttivo
- **Incremento**
  - Procedere per aggiunte successive a un impianto base (scultura): solo costruttivo
- **Prototipo**
  - Per 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



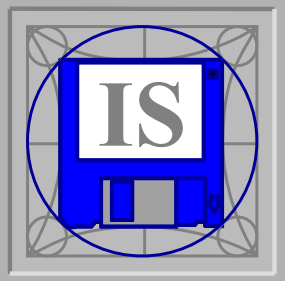
Modello sequenziale (a cascata) – 1

- **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

- L'ingresso in uno stato è vincolato da pre-condizioni (*gate*) che sono le post-condizioni delle corrispondenti transizioni
  - Soddisfatte in modo documentale e solo in fine tramite dimostrazione del prodotto SW
- Fasi distinte e non sovrapposte nel tempo
- Modello adatto allo sviluppo di sistemi complessi, anche sul piano organizzativo



Modello sequenziale (a cascata) – 3

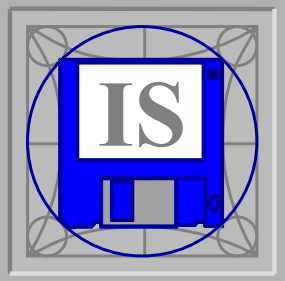
- Ogni fase (stato o transizione) viene definita in termini di
  - Attività previste e prodotti attesi in ingresso e in uscita
  - Contenuti e struttura dei documenti
  - Responsabilità e ruoli coinvolti
  - Scadenze di consegna dei prodotti
- Entrare, uscire, stazionare in una fase comporta lo svolgimento di determinate azioni
  - Realizzate come attività erogate da 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, ...)

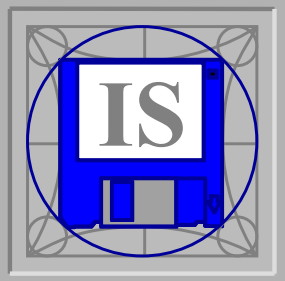
Legenda
AR: analisi dei requisiti
DA: progetto (*design*) architetturale
DD: progetto (*design*) di dettaglio

La manutenzione comporta ritorni sulle fasi precedenti



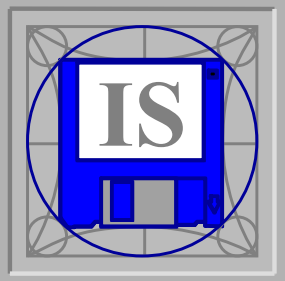
Critica del modello sequenziale

- **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: cascata con ritorni**
  - **Ogni ciclo di ritorno raggruppa sotto-sequenze di fasi**



Ritorni: iterazione o incremento?

- Non sempre gli *stakeholder* hanno le idee chiare
  - Come aiutarli: iterando o incrementando?
- Problemi particolarmente complessi chiedono di procedere a tentoni
  - Tramite iterazioni potenzialmente distruttive
- Altrimenti meglio procedere per piccoli passi incrementali
  - Posticipare l'integrazione delle parti del sistema è rischioso
  - Se le cose vanno male, restiamo con niente in mano: *big-bang integration*
  - Meglio l'integrazione continua
- Iterazione e incremento coincidono quando la sostituzione raffina ma non ha impatto sul resto



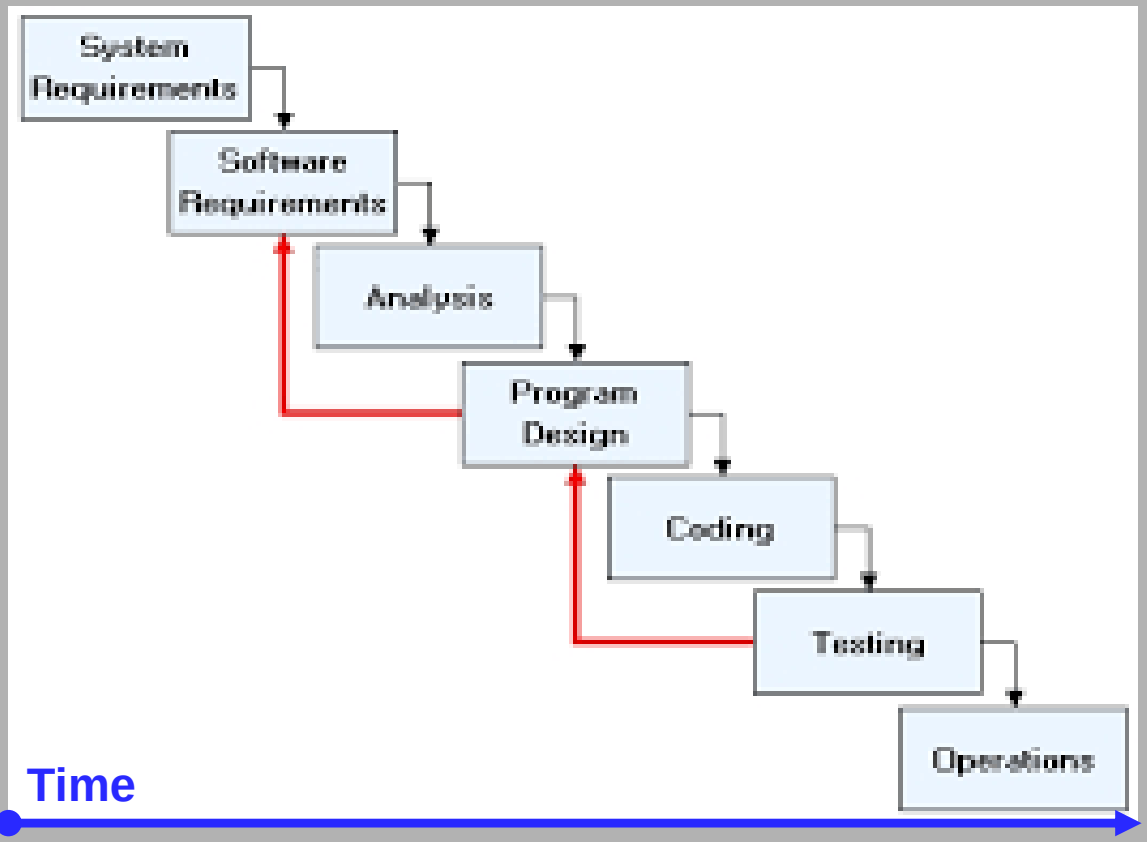
Vantaggi dei modelli incrementali

- Possono produrre valore a ogni incremento
  - Un insieme di funzionalità diventa presto disponibile
  - I primi incrementi possono essere frutto di prototipazione, aiutando a fissare meglio i requisiti per gli incrementi successivi
- Ogni incremento riduce il rischio di fallimento
  - Senza però azzerarlo …
- Le funzionalità fondamentali vanno sviluppate prima
  - Così che esse siano più frequentemente verificate così diventando via via più stabili

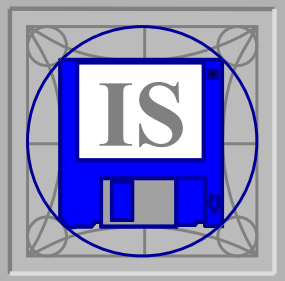
Vantaggi dei modelli iterativi

- Applicabili a qualunque modello di sviluppo
  - Ma comportando forte potenziale distruttivo
- Consentono maggior capacità di adattamento
  - Insorgere di problemi, cambio di requisiti, collasso tecnologico
- Ma comportano il rischio di non convergenza
- Soluzione generale
  - Decomporre la realizzazione del sistema
  - Identificare e lavorare prima sulle parti più critiche: quelle più complesse o quelle con i requisiti più incerti
  - Limitando superiormente il numero di iterazioni

Rischi dei modelli iterativi



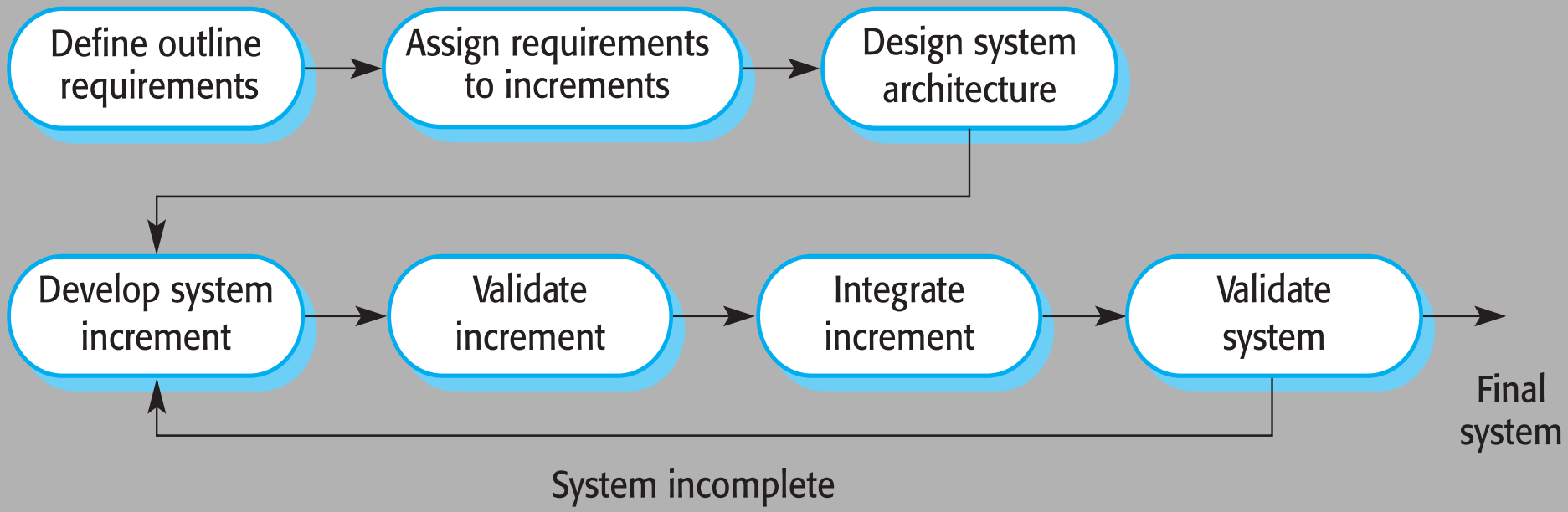
Ogni iterazione comporta un ritorno all'indietro nella direzione opposta all'avanzamento del tempo



Modello incrementale – 1

- 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
  - 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

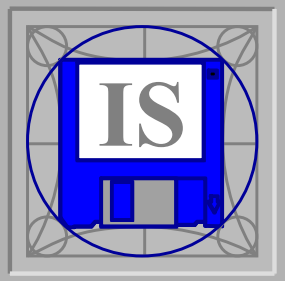
Schema generale



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.



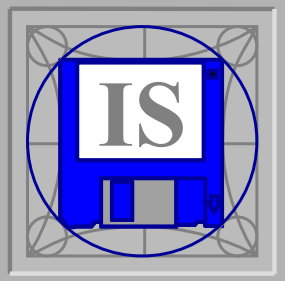
Modello incrementale – 2

- **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**
  - **Per decidere preventivamente il numero di incrementi**
    - Assegnando specifici obiettivi a ciascuno

Di questo parleremo ampiamente più avanti

- **La realizzazione è incrementale**
  - **L'analisi dei requisiti può essere raffinata – preservando l'architettura – insieme alla progettazione di dettaglio**
  - **Il completamento dei primi incrementi deve rendere disponibili le principali funzionalità**

Schema secondo ISO 12207:1995



Modello evolutivo – 1

- Aiuta a rispondere a bisogni non inizialmente preventivabili
- Può richiedere il rilascio e il mantenimento di più versioni esterne attive in parallelo
- Comporta il riattraversamento di più stati di ciclo di vita

Modello evolutivo – 2

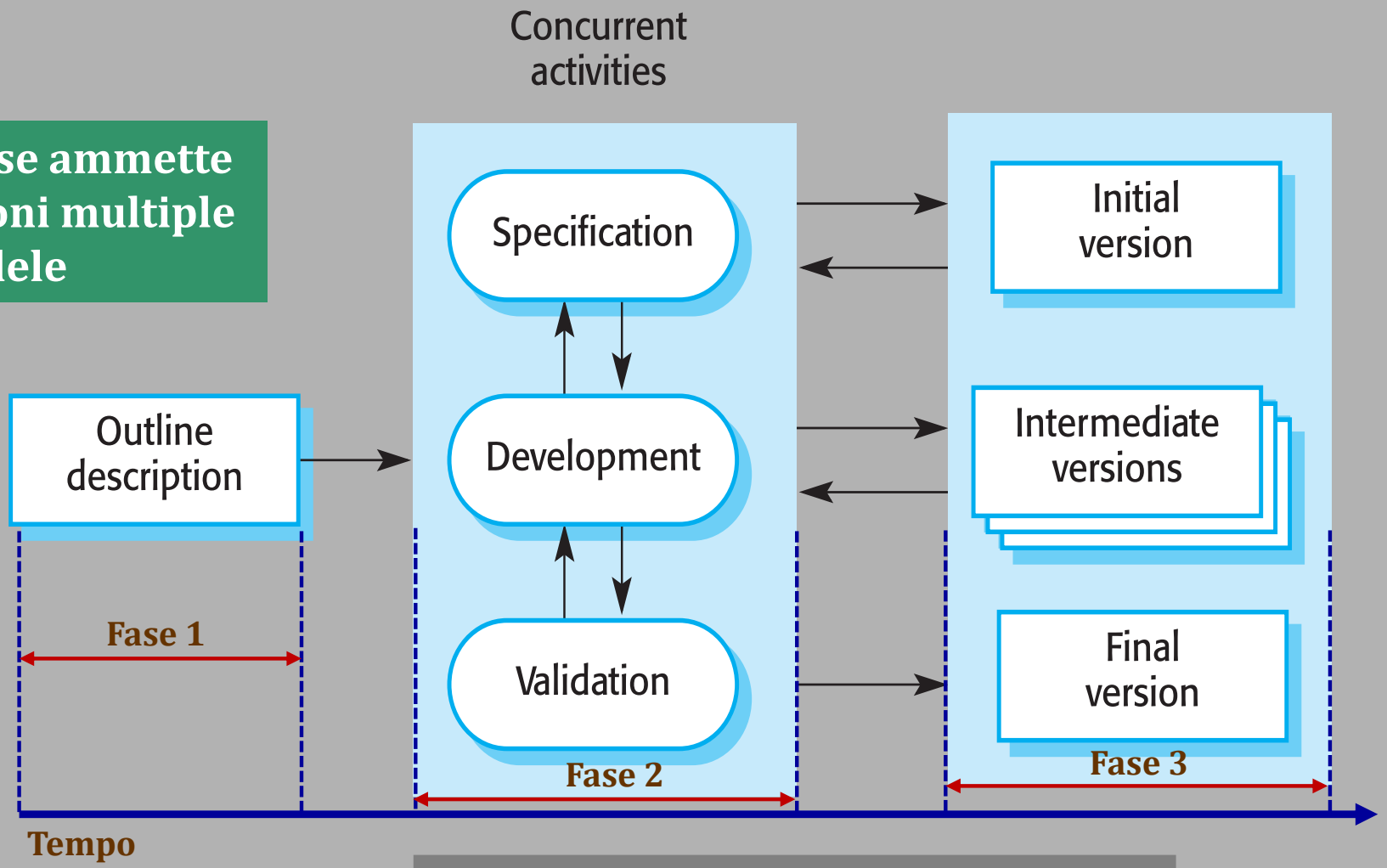
- Analisi preliminare
  - Identificare i requisiti fondamentali
  - Definire l'architettura di base, massimamente modulare per facilitare evoluzione futura
  - Pianificare i passi di analisi e realizzazione successivi, anche sovrapponendoli tra loro
- Analisi e realizzazione di una singola evoluzione raffinano ed estendono lo stato precedente
- Ogni evoluzione rilascia versioni indipendenti
  - Fintanto che necessario

Firefox is shipped using a train **model**. Without going into too much details, this means that we maintain several channel in parallel (Nightly, Beta, **Release** and ESR). Normal changes happen in Nightly. When a change needs to be cherry-picked from Nightly to another branch, the process is called "Uplift". Jan 21, 2019

release.mozilla.org

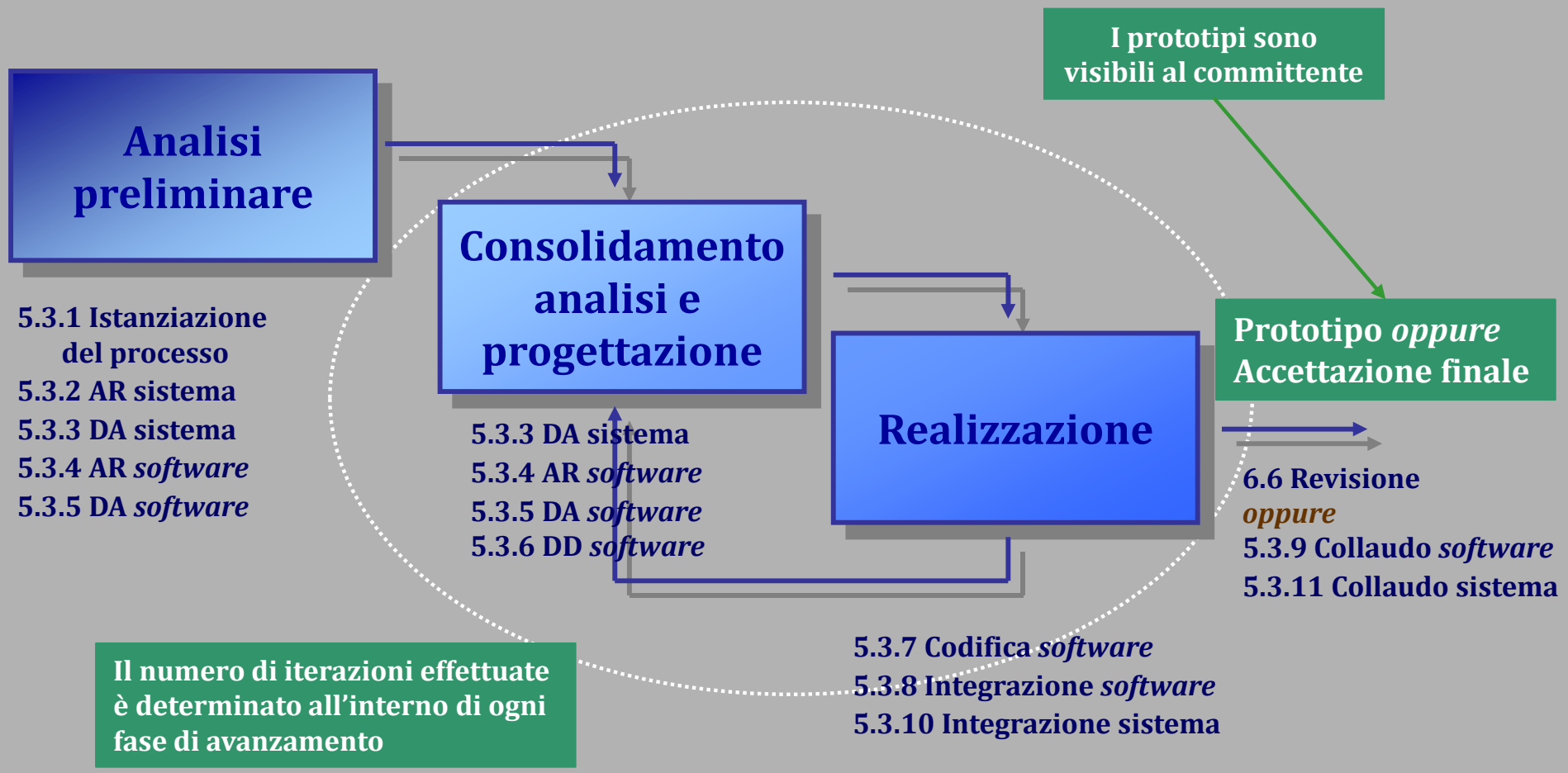
Schema generale

Ogni fase ammette iterazioni multiple e parallele

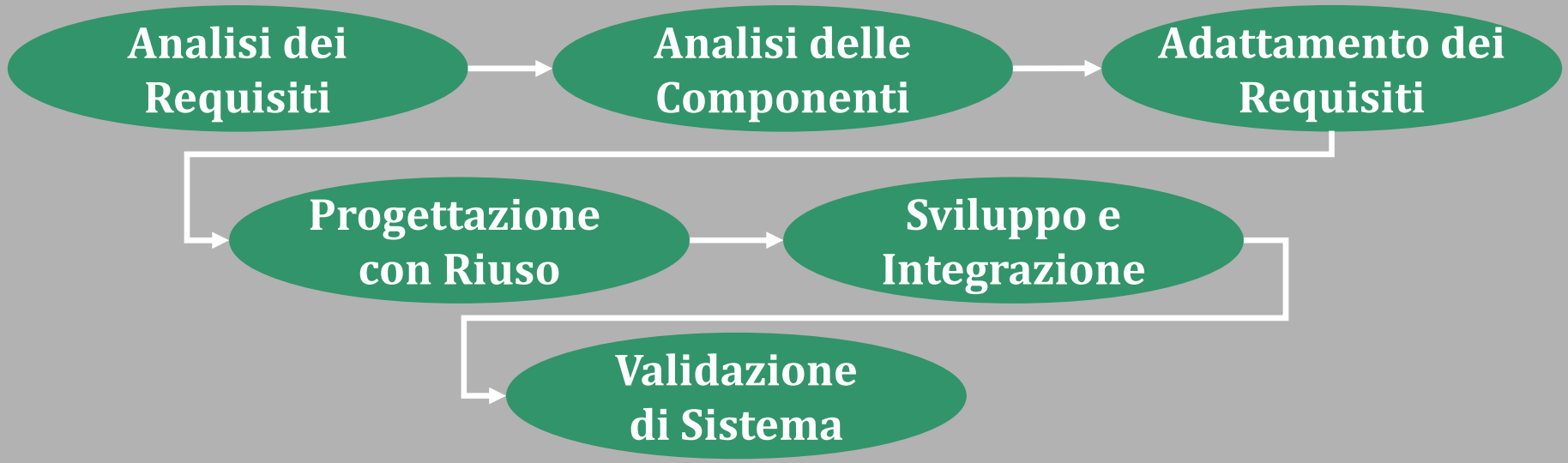


Tratto da: Ian Sommerville, *Software Engineering*, 8th ed.

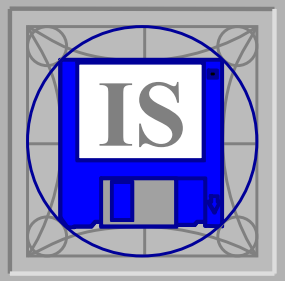
Schema secondo ISO 12207:1995



Modello a componenti

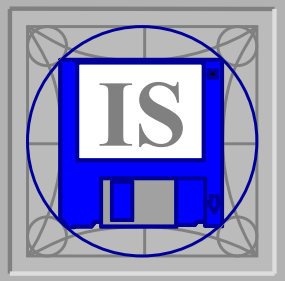


- Molto di quello che ci serve fare è già stato fatto e molto di quello che faremo ci potrà servire ancora
  - Analisi dei requisiti guidata dalla possibilità di riuso di quanto già esista
  - Realizzazione che cerca di favorire riuso futuro



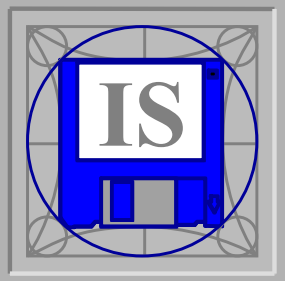
Metodi agili – 1

- **Nascono alla fine degli ’90 come reazione alla eccessiva rigidità dei modelli allora prevalenti**
  - **http://agilemanifesto.org/**
- **Si basano su quattro principi fondanti**
  - ***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



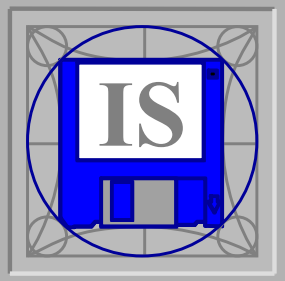
Contro-argomentazioni

- 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

- L'idea base è il concetto di "*user story*"
  - Una funzionalità significativa che l'utente vuole realizzare con il SW richiesto
- Ogni "*user story*" è definita da
  - Un documento di descrizione del problema individuato
  - La minuta 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

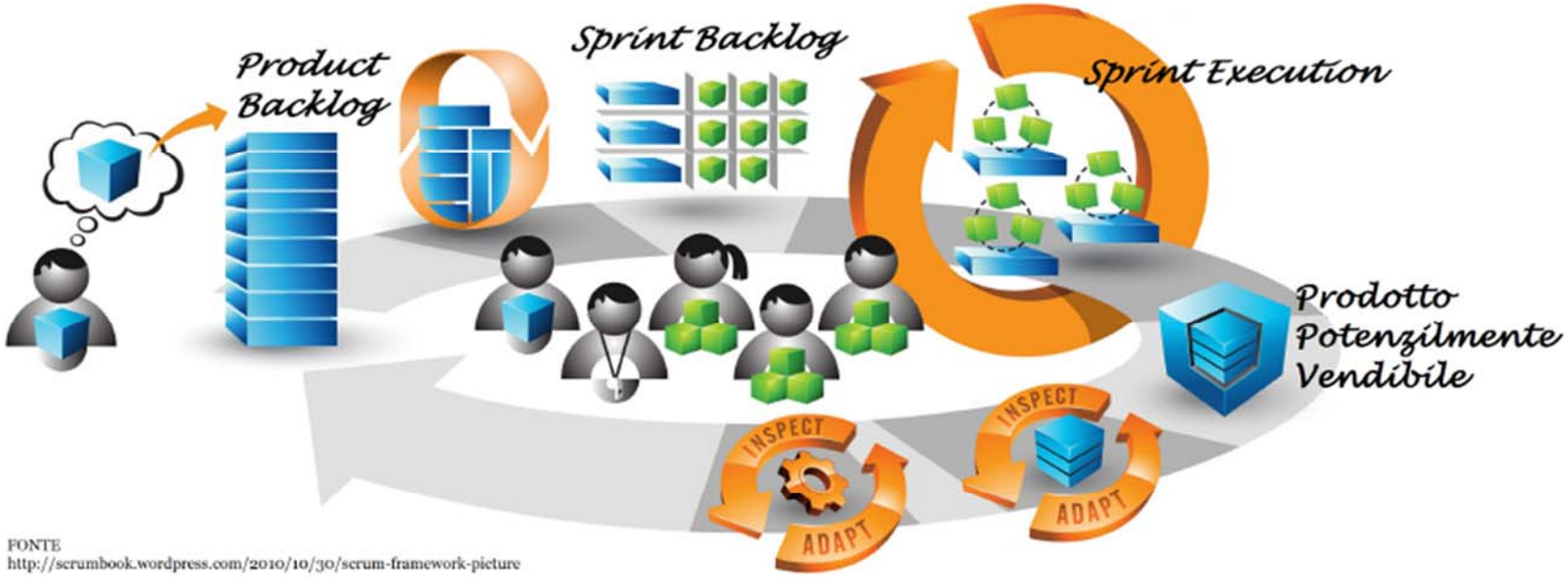


Metodi agili – 3

- **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 (organizzazione dietro caos apparente), Kanban (*just-in-time*), Scrumban

Il modello incrementale è il paradiso dell'agile

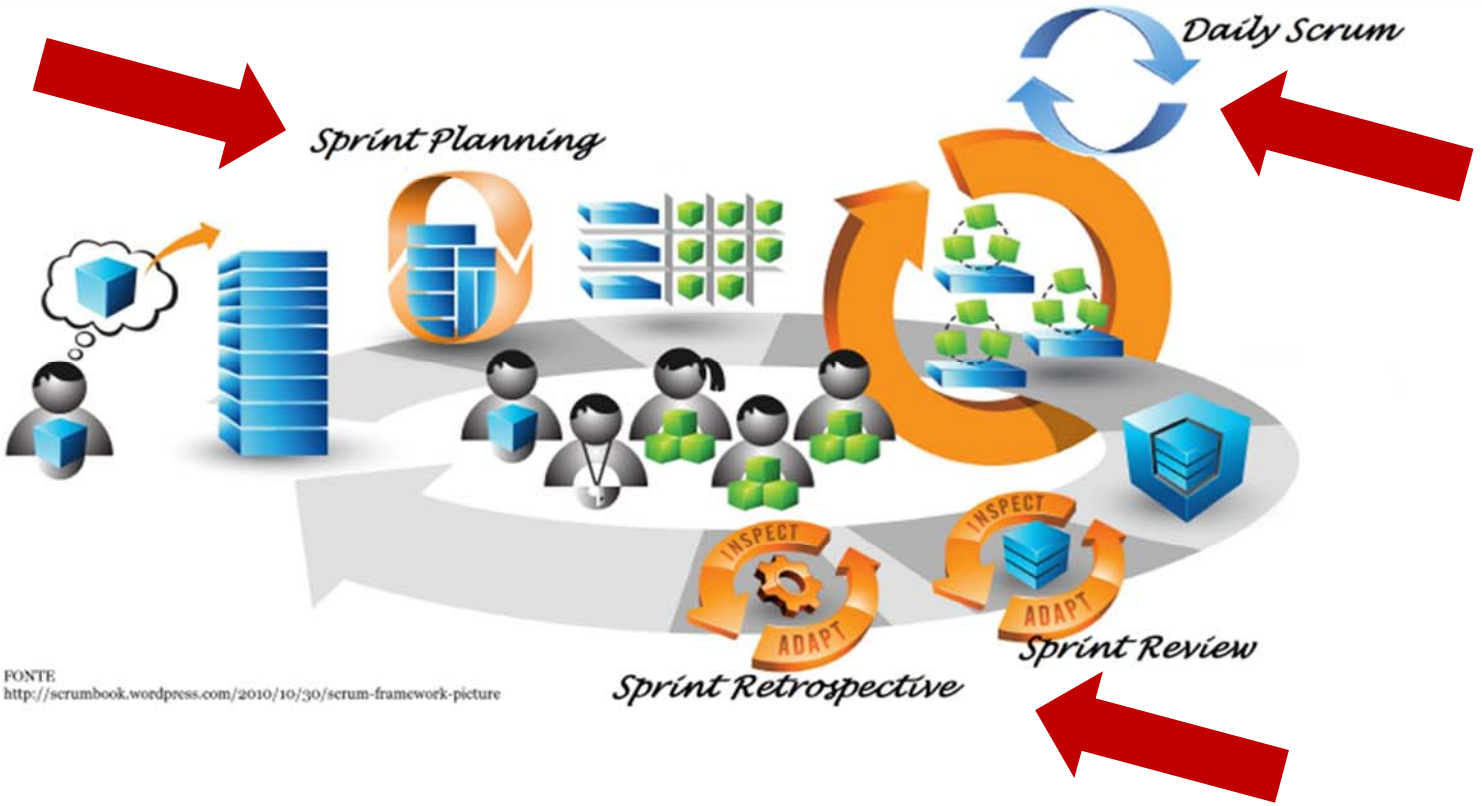
Scrum – 1



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

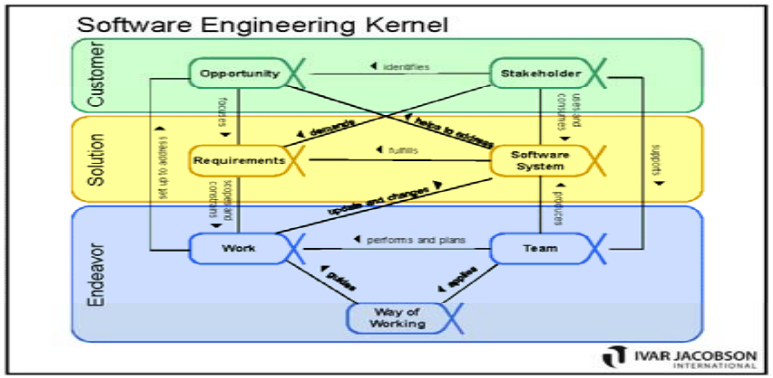


FONTE
http://scrumbook.wordpress.com/2010/10/30/scrum-framework-picture

- **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

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.

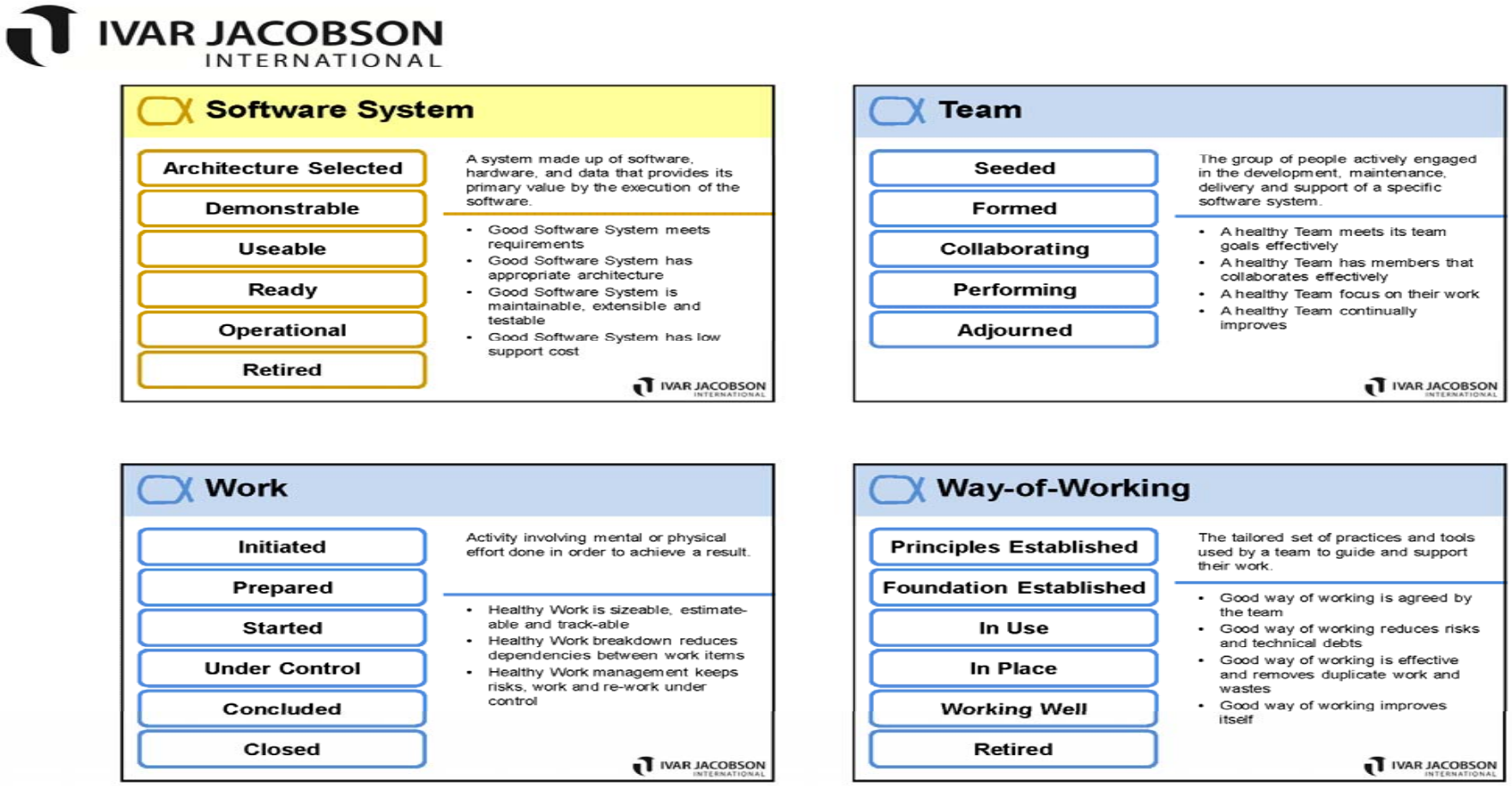
- Good Requirements meets real needs
- Good Requirements has clear scope
- Good Requirements are coherent and well organized
- Good Requirements help drive development

IVAR JACOBSON INTERNATIONAL

www.ivarjacobson.com/semat

Sheet 1 of 2

Il ciclo di vita secondo SEMAT /2

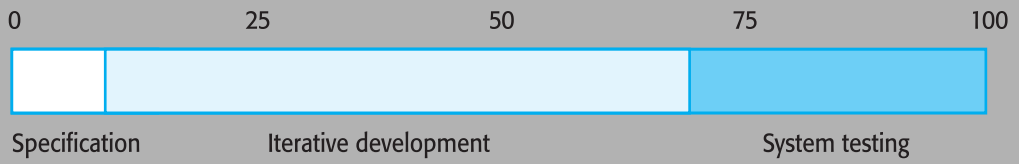


Ripartizione dei costi nei modelli

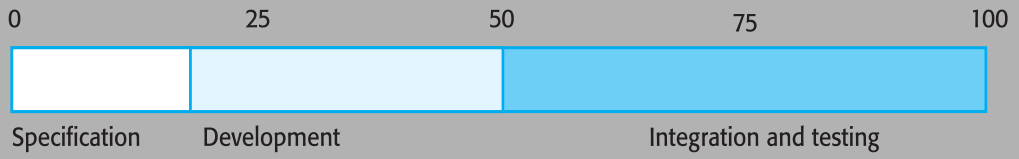
- Applicazioni "normali"
  - ~ 60% → realizzazione
  - ~ 40% → qualifica
- I costi complessivi variano al variare del dominio e del tipo di sistema
- La ripartizione dei costi sulle fasi varia al variare del modello e del dominio
  - Sistemi critici: > 60% qualifica

Modello sequenziale

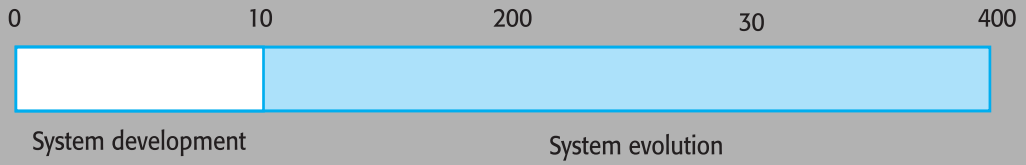
Modello iterativo



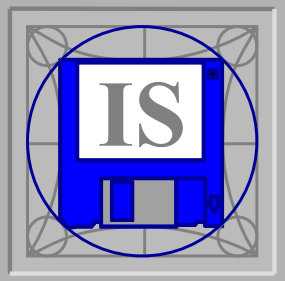
Modello "*component-based*"



Modello evolutivo



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



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