



UNIVERSITÀ DEGLI STUDI
DI MILANO

Lezione 2

Processo di produzione del software

- Riconoscimento che:
 - produrre software non è *solo* scrivere codice
 - bisogna risolvere problema **comunicazione**
 - bisogna essere **rigorosi**

Bauer-Zemanek's Hypothesis (H3):

Formal methods significantly reduce design errors, or eliminate them early

- ci sono tanti **aspetti** da considerare (uno alla volta)

Modellare ciclo di vita del sw

- Identificare vari passi e attività necessarie
 - precedenze temporali
 - soggetti diversi
- Porsi due domande
 - Cosa devo fare adesso?
 - Fino a quando?

Studio di fattibilità

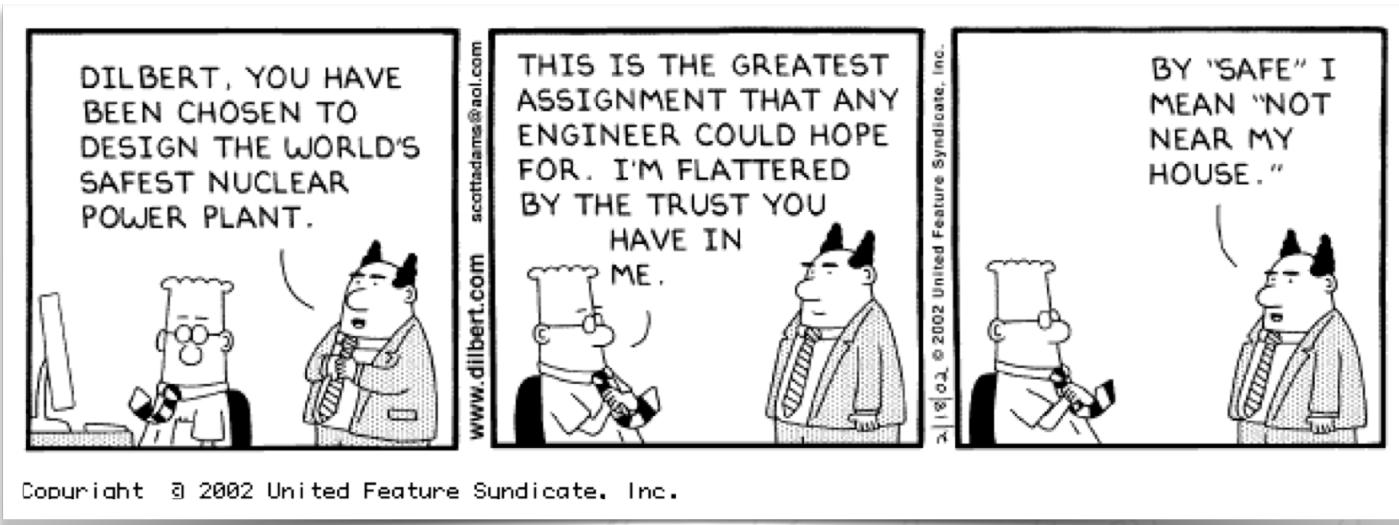
- Definizione preliminare del problema
 - possibile mercato e concorrenti
- Studio di diversi scenari di realizzazione
 - scelte architetturali e hardware necessario
 - sviluppo in proprio, subappalto
- Stima dei costi, tempi di sviluppo, risorse necessarie e benefici delle varie soluzioni
- Spesso difficile fare una analisi approfondita
 - spesso viene commissionata all'esterno
 - si hanno limiti di tempo stringenti
- **POSSIBILI OUTPUT:** un documento spesso in linguaggio naturale

Analisi e specifica dei requisiti

- Comprendere il dominio applicativo
- Identificare gli stakeholders
- Quali sono le funzionalità richieste?
 - **Cosa** deve fare il sistema?
 - Non interessa il **come...**
 - È il punto di vista dell'utente, non gli interessa il progetto o l'implementazione
 - Quali sono le altre qualità richieste?

Requirements e stakeholders

<https://dilbert.com/strip/2002-02-18>



Requirements analysis and specifications

POSSIBILI OUTPUT:

- Documento di specifica
 - documento contrattuale approvato dal committente
 - base per il lavoro di design e verifica
 - importanza di avere documento formale
- Manuale utente o maschere di interazione
 - vista esterna per eccellenza
- Piano dei test di sistema
 - collaudi che certificano correttezza

Davis' Law (L4):

The value of models depends on the view taken, but none is best for all purposes

Progettazione (Design)

- Come le specifiche precedentemente trovate possono essere realizzate in maniera opportuna?
- Definizione dell'architettura del sistema
 - Scelta di una architettura software di riferimento
 - Scomposizione in moduli o oggetti
 - Identificazione di patterns

POSSIBILI OUTPUT:

- documento di specifica di progetto
 - diversi linguaggi



Programmazione e test di unità

- Le *scatole nere* definite al punto precedente vengono realizzate
- I singoli moduli vengono testati indipendentemente
 - framework per il test
 - moduli stub (fittizi)
 - moduli driver (guida)

Coding and module/unit testing

POSSIBILI OUTPUT

- un insieme di moduli sviluppati separatamente ...
- ... con interfaccia concordata ...
- ... singolarmente verificati

Integrazione e test di sistema

- Vengono uniti i vari componenti
- **test di integrazione:** sottoinsiemi dell'applicazione: vengono sostituiti mano a mano i moduli di testing con i moduli reali
 - top-down
 - o bottom-up?

Manutenzione

... ne abbiamo già parlato

- Correttiva
- Adattativa
- Perfettiva

OUTPUT: un prodotto migliore (per non dire corretto)

Altre attività

- Documentazione
 - Può essere vista come attività trasversale
 - Nella pratica spesso è una attività a posteriori
- Verifica e controllo qualità
 - QA: quality assurance
- Gestione del processo
 - Gestione incentivi, responsabilità, eccezioni
- Gestione configurazioni
 - Può essere vista come relazioni inter-progettuali

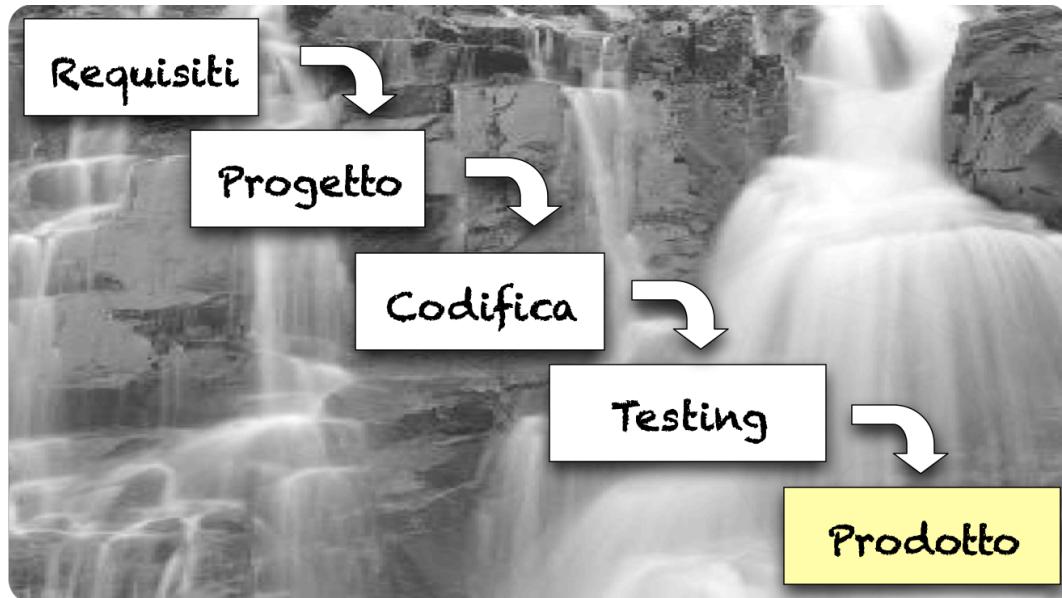


UNIVERSITÀ DEGLI STUDI
DI MILANO

Modelli di ciclo di vita del software

Modello a cascata

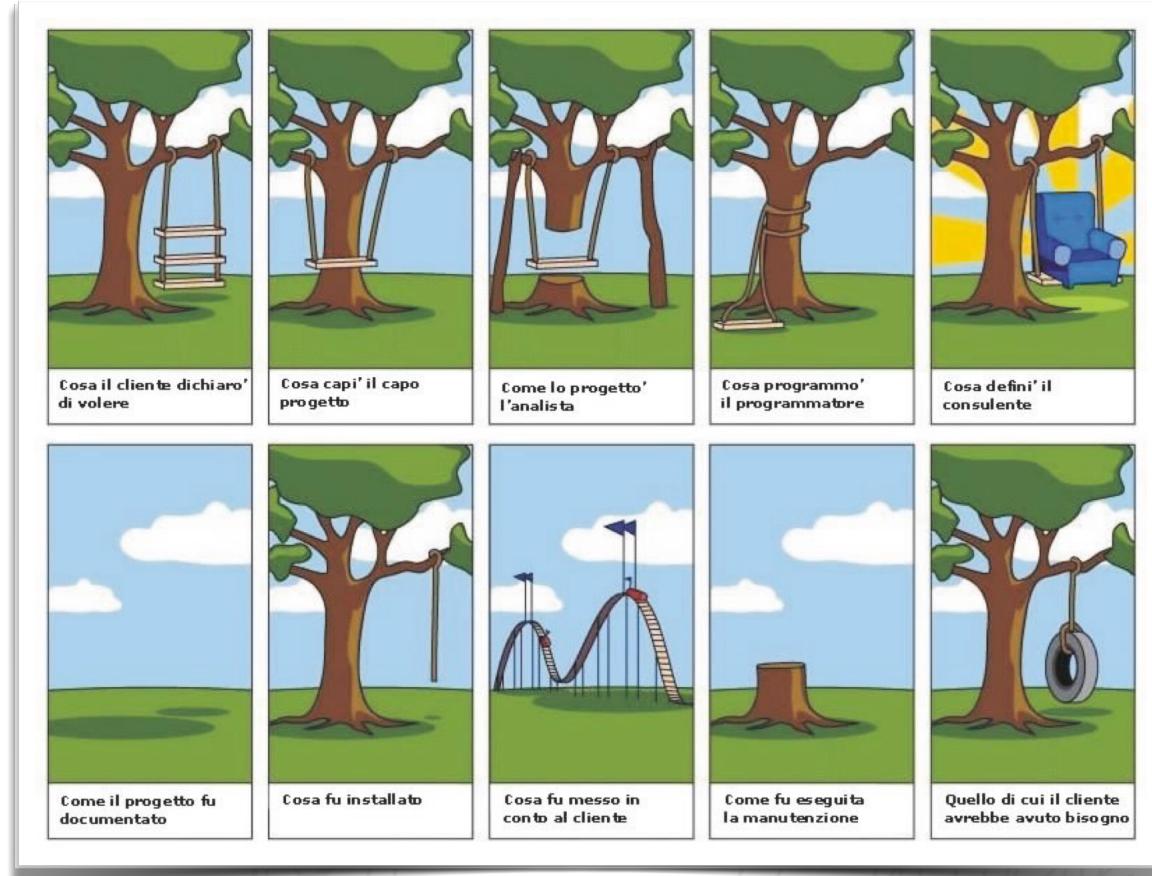
- Fine degli anni '50
- ... ma diventato "famoso" negli anni '70
- una famiglia di processi



Modello a cascata

- Forza una **progressione lineare** da una fase alla successiva
- Le varie fasi comunicano attraverso semilavorati
 - permette quindi una **separazione dei compiti**
- È possibile fare una **pianificazione dei tempi** e monitoring dello stato di avanzamento
 - ma a senso unico

Problemi di comunicazione



Manutenzione

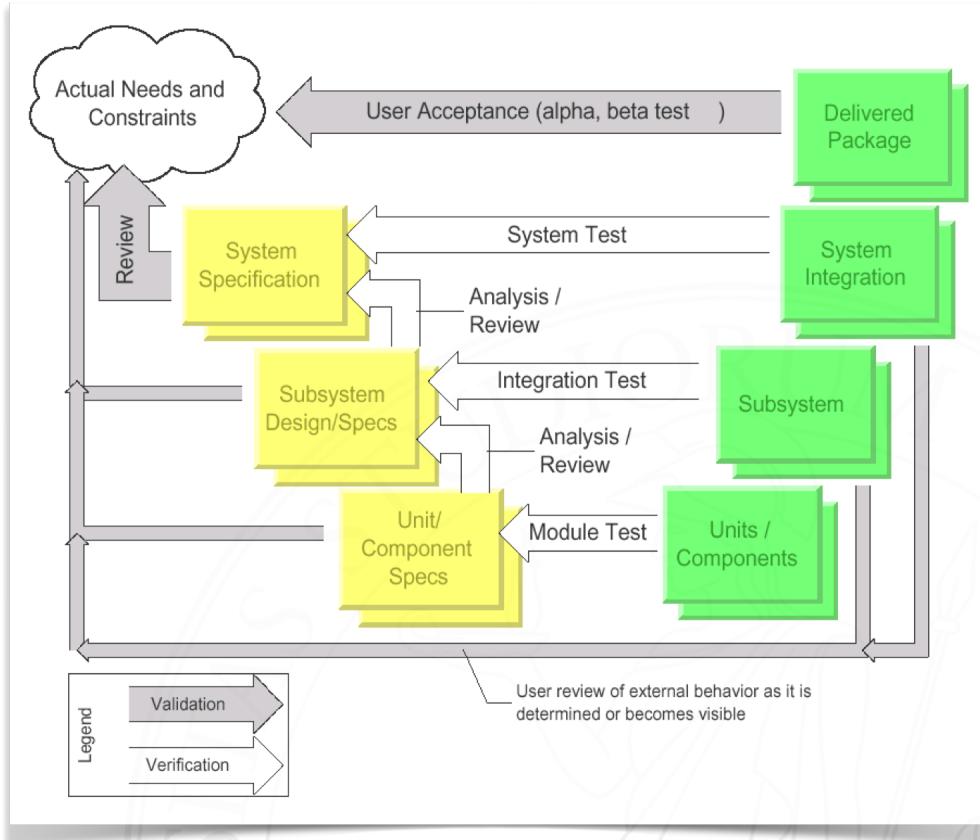
- È una fase prevista?
 - È considerata una eccezione... senz'altro non pianificata
- È possibile farla nel modello a cascata?
 - Non si può tornare indietro!!
 - Niente modifiche a specifiche, codice, documentazione...
- ... ma il 70% dei costi di sviluppo ricadono in questa fase

Problemi

- Rigidità
- Congelamento sotto prodotti
 - specifiche e stime fatte solo nelle prime fasi
- Monoliticità
 - Tutta la pianificazione è orientata ad un singolo rilascio
 - manutenzione fatta solo sul codice

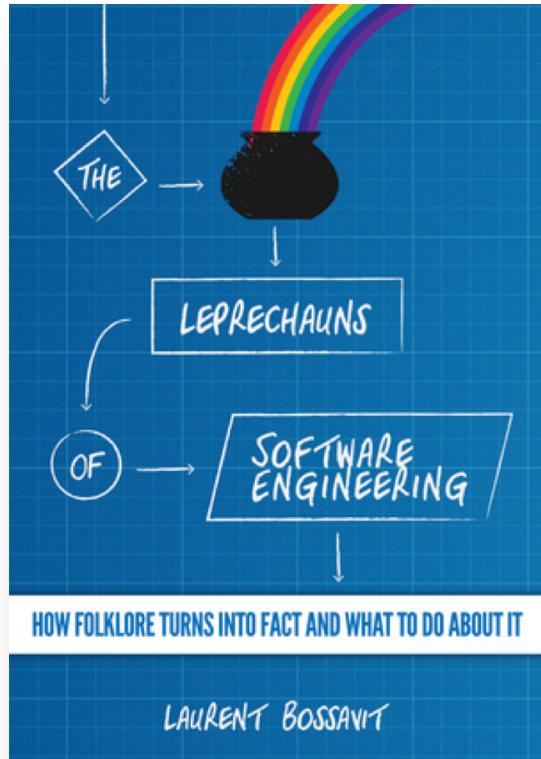
Riposizionamento modello a V

Espande
fase di
testing e
chiarisce
alcune
relazioni



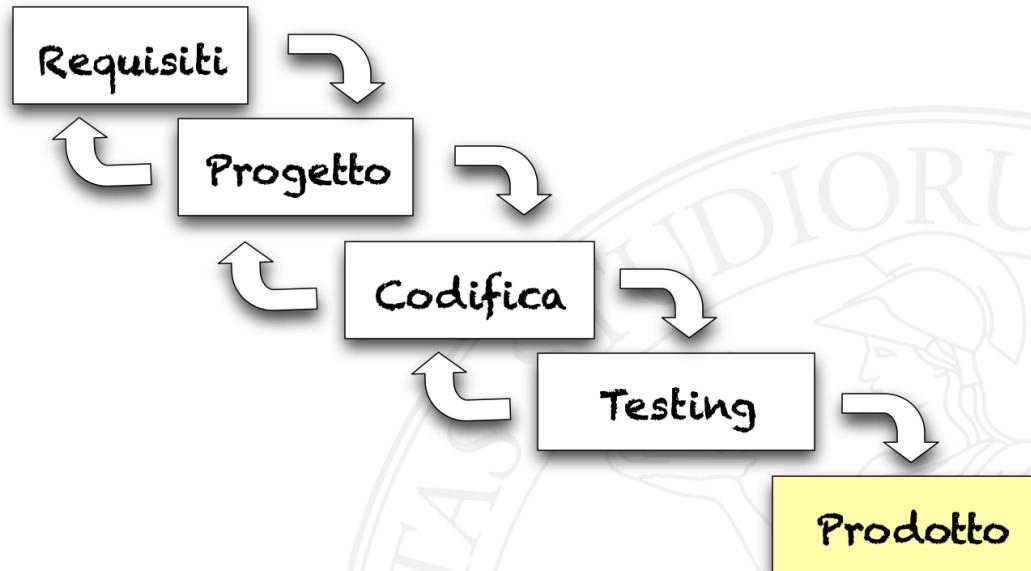
Who's Afraid of The Big Bad Waterfall?

<https://leanpub.com/leprechauns>



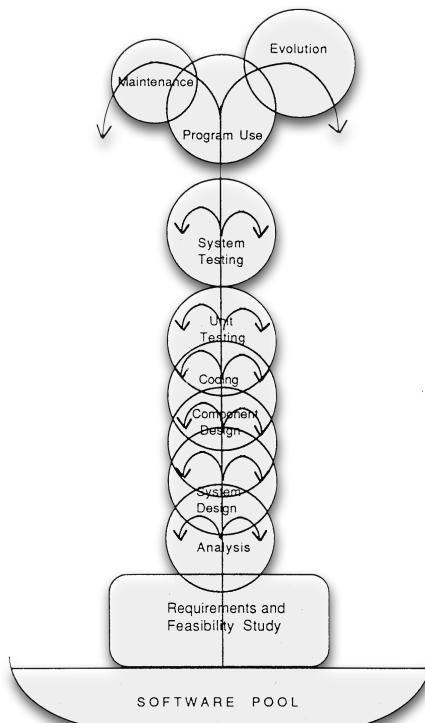
Varianti del processo a cascata

- Con singola retroazione
 - una fase può portare a modifiche nella fase precedente (**iterazione**)



Modello ciclo di vita a fontana

Henderson-Sellers 1993



- con "da capo"
- imprevedibile la profondità di ogni zampillo
- se guardiamo però in cima abbiamo che siamo arrivati a una versione **incrementale**

Modelli iterativi vs. incrementali

- Si parla di **incrementale** quando nelle **iterazioni** viene inclusa la consegna

Ad esempio si può parlare di:

- Implementazione **iterativa**
 - **stressa la modularizzazione e l'identificazione di sottosistemi e si ripetono fasi di coding ed integrazione**
- Sviluppo **incrementale**
 - viene esteso a tutte le fasi (specifiche comprese)
 - il “modello a cascata” viene eseguito per ogni incremento
 - la fase di manutenzione in senso stretto sparisce diventando semplice riciclo

Modelli prototipali

- Prototipo usa e getta (*throw away*)
- Pubblici o esterni
 - Per capire meglio i requisiti e/o far compiere scelte all'utente interfacce
- Privati o interni
 - Per esplorare nuovi strumenti, linguaggi, diverse scelte per problema difficili progetti pilota, testbed

Boehm's Law (L3)

Prototyping (significantly) reduces requirements and design errors,
especially for user interfaces

“Problemi” con modelli incrementali

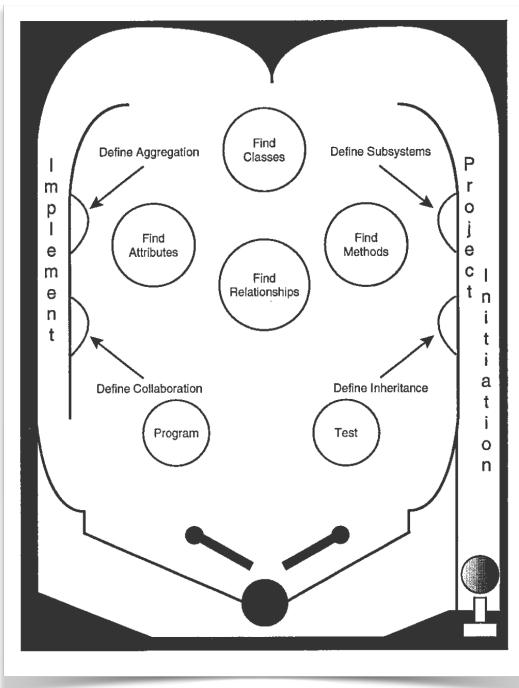
<https://bit.ly/3SYYs8y>

From Waterfall to Iterative Development – A Challenging Transition for Project Managers

- Viene complicato il lavoro di planning
 - Lo stato del processo è meno visibile
 - Bisogna pianificare tutte le iterazioni
- Si riconosce che bisogna rimettere mano a ciò che si è fatto
 - Il sistema può non convergere
- Cosa è una iterazione, quanto dura?
 - Rischio di voler mettere troppo nella prima iterazione
 - Rischio di overhead dovuto a troppe iterazioni
 - Rischio di avere un eccessivo overlapping tra le iterazioni
 - Non si ha tempo di avere feedback dell'utente

Pinball Life-Cycle

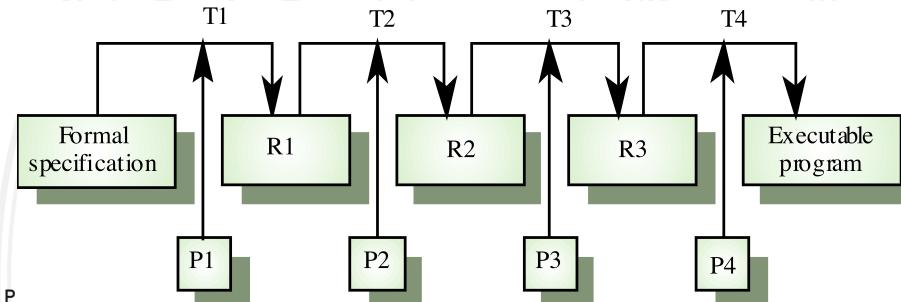
Ambler, 1994



- Indefinito
- Qualunque passo è possibile
- Non vengono imposti vincoli temporali
- non misurabile

Modelli trasformazionali

- Raffinamenti di rappresentazioni formali del problema
 - ad ogni passo vengono specializzate, ottimizzate, rese più concrete attraverso trasformazioni automatiche o comunque controllate
- A partire da specifiche formali si ottiene un prototipo
 - differisce dal prodotto finale per efficienza e completezza
 - passi di trasformazioni (formalmente dimostrabili come corretti) portano ad avere la versione finale

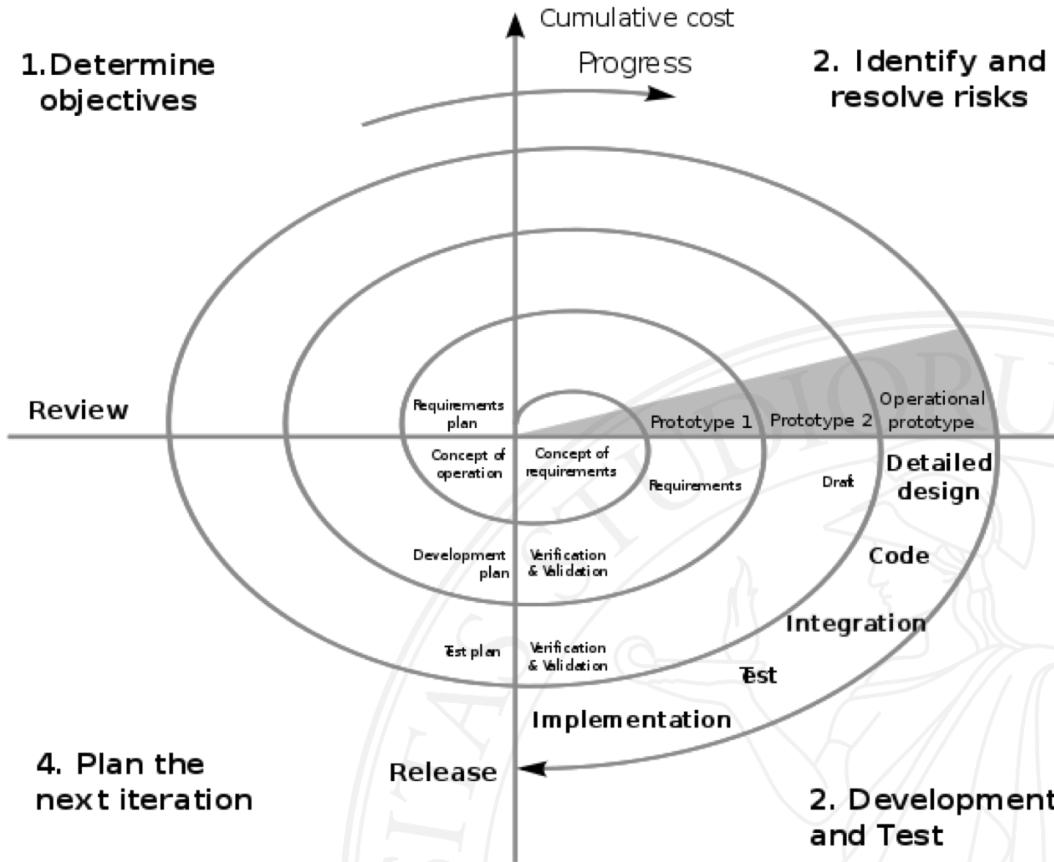


MetaModello a spirale

Boehm 1988

- Non è un modello particolare, ma un framework in cui si possono inquadrare altri modelli
- Guidato dall'analisi dei **rischi**
- Fasi:
 - Determinazioni obiettivi, alternative e vincoli
 - Valutazione alternative, identificazione rischi
 - Sviluppo e verifica
 - Pianificazione fase successiva

Modello a spirale

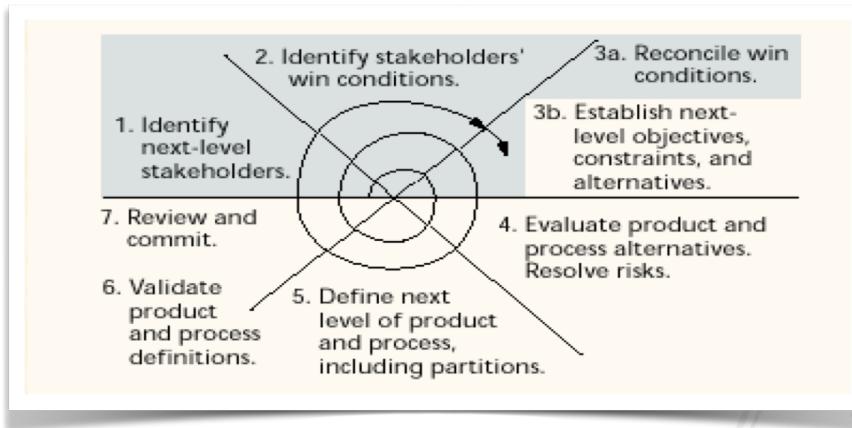


Modello Spirale Win-Win

<https://bit.ly/3ryt5WB>

Boem 1998

- Evidenzia le comunicazioni con i clienti e che non sono di tipo “pacifico” ma necessitano contrattazioni e negoziazione.

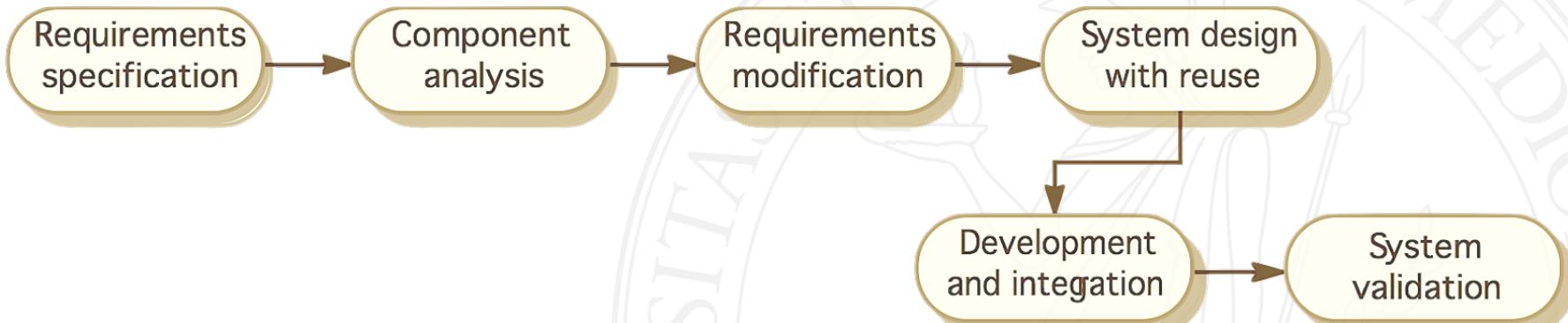


Win Win vuol dire che entrambi le parti vincono (o hanno l'illusione di vincere)

Modello COTS

Component Off The Shelf

- Partire dalla disponibilità interna o sul mercato di moduli preesistenti e creare il sistema a partire da quelli mediante integrazione
- Apre nuovi problemi (tematiche) quali:
 - classificazione dei moduli
 - retrieval dei moduli





UNIVERSITÀ DEGLI STUDI
DI MILANO

Metodologie Agili

eXtreme Programming

Il manifesto dei metodi agili

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.

Kent Beck
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler

James Grenning
Jim Highsmith
Andrew Hunt
Ron Jeffries
Jon Kern
Brian Marick

Robert C. Martin
Steve Mellor
Ken Schwaber
Jeff Sutherland
Dave Thomas