Introduzione alla data science e al pensiero computazionale
Lezione 5: Il processo di produzione del software

Giancarlo Succi
Dipartimento di Informatica – Scienza e Ingegneria
Università di Bologna
g.succi@unibo.it

Indice

- Motivazione
- La crisi del software
- Problemi *Tame* e *Wicked*
- Controllare il processo
- Meccanismi di coordinamento
- Modello di sviluppo a cascata
- Modello di sviluppo a spirale
- Il manifesto Agile

Ariane 5 (1996)

Software Engineering (1/2)

Software Engineering o Ingegneria del Software

Ingegnerizzare la produzione del software

Spesso si fanno analogie tra la produzione del software e cucinare...

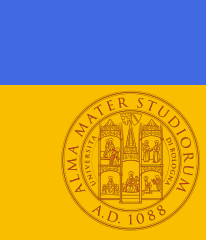
Che lo show abbia inizio!

Signore e signori: Charlie Chaplin

Sondaggio

Sulla base del video, qual è lo scopo dell'ingegneria dle software?

Idee



La mia idea dell'ingegnere del software

Video on Dr. House

Lean software development o approccio lean alla produzione del software

Lean software development o approccio lean alla produzione del software

Premesse sul Lean Management

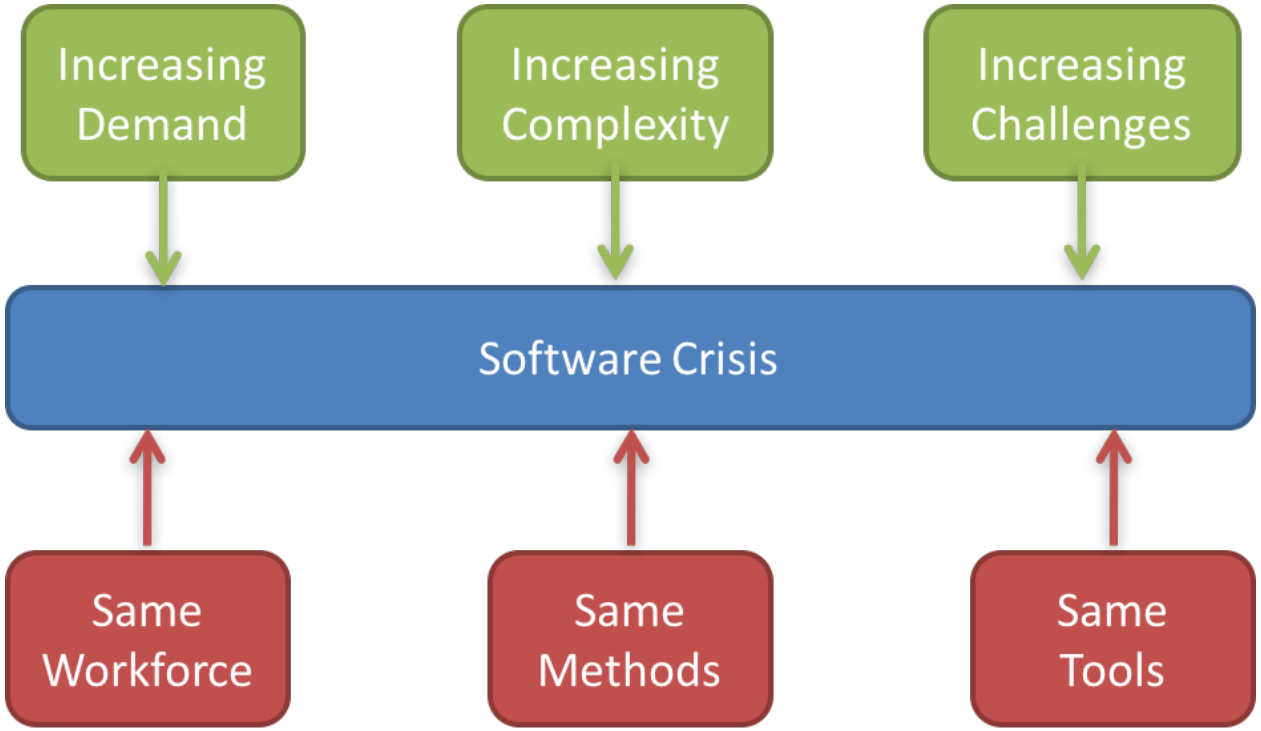
La “Software Crisis” (1/3)

La “Software Crisis” o “crisi del software” è il fenomeno dovuto all’aumento della potenza dei calcolatori: in termini *terra terra*, finché i calcolatori erano molto piccoli, la programmazione non presentava problemi, quando le macchine hanno iniziate a crescere, i problemi hanno iniziato a comparire, e ora che abbiamo computer giganteschi, la programmazione è pure diventata un problema gigantesco.
Edsger Dijkstra, **1972**

http://qualityandprogramming.blogspot.ru/2012/03/crisis-del-software.html

La “Software Crisis” (2/3)

Sequenze di rivoluzioni tecnologiche si sono viste in molte aree dello scibile umano grazie al software.



Purtroppo, l’ingegneria del software non ha avuto una analoga rivoluzione!

http://www.slideshare.net/rui_curado/abse-and-atomweaver-a-quantum-leap-in-software-development

La “Software Crisis” (3/3)

Le cause della crisi del software sono collegate alla complessità crescente di hardware e software associato, e tale crisi si è manifestata in una moltitudine di modi:

- Progetti che sforano il budget
- Progetti che non rispettano i tempi
- Sistemi software inefficienti
- Sistemi software di bassa qualità
- Sistemi software che non rispndono a requisiti
- Progetti diventati ingestibili
- Codice software illeggibile e impossibile da mantenere
- Software mai consegnato al cliente

https://en.wikipedia.org/wiki/Software_crisis

Esempi di errori dei sistemi software (1/2)

NASA's Mars Climate Orbiter – Nella rotta verso Marte nel 1998 il Climate Orbiter spacecraft si 'è perso nello spazio. Anche se inizialmente non si capì la ragione dell'errore, alla fine si trovò che un subcontraente commise un banale errore di trasformazioni delle misure dal sistema inglese a quello metrico.

Questa svista imbarazzante spedì la navetta spaziale da **125 milioni di dollari** fatalmente vicina alla superficie di Marte dopo aver tentato di stabilizzare la sua orbita troppo in basso.

Estratto con modifiche da: https://raygun.com/blog/2014/05/10-costly-software-errors-history/

Esempi di errori dei sistemi software (2/2)

Ariane 5 Flight 501 – L'allora nuovissimo satellite europeo riutilizzò un software dal suo predecessore, l'Ariane 4. Sfortunatamente, l'Ariane 5 aveva un motore più veloce e questo rivelò un baco precedentemente passato inosservato. Dopo 36 secondi dal lancio, i progettisti inziarono l'operazione di autodistruzione a seguito di una serie di errori dei computer. In sintesi, il software stava provando a inserire un numero a 64 bit in uno spazio di 16 bit, con il risultato di mandare in tilt sia i computer primari che quelli ausiliari, visto che entrambi usavano lo stesso software.

La costruzione dell'Ariane 5 era costata **8 billioni di dollari** e stava trasportando un satellite da **500 millioni di dollari**.

Estratto con modifiche da: https://raygun.com/blog/2014/05/10-costly-software-errors-history/ Video: https://youtu.be/qnHn8W1Em6E

Ancora sulla “Software Crisis” (1/2)

Le problematicità nello sviluppo software proviene dalla nostra intrinsica limitazione nel comprenderlo, e dalle difficoltà strutturali, che possono essere divise in quattro categorie:

- **Complessità:** i sistemi software consistono di parti multiple e diverse che possono essere in molti stati diversi. Questo li rende difficile ad essere concepiti, descritti e controllati.
- **Conformità:** molto spesso il software deve essere integrato con altro software sviluppato precedentemente in contesti e con regolamentazioni che possono essere diverse. Tutti queste condizioni e vincoli crescono ed evolvono nel tempo e rendono anche molto probabile che il software debba essere cambiato in futuro.

Ancora sulla "Software Crisis" (2/2)

- **Modificabilità:** tutto il software di successo è soggetto a modifiche – i clienti e gli utenti di tale software scoprono nuovi possibili usi e quindi chiedono tali modifiche. Inoltre, il software di successo sopravvive ai cicli di vita dell'ambiente circostante come quelli dei sistemi operativi, dell'hardware ecc. Questo implica che il software debba essere aggiornato ai nuovi ambienti dove deve funzionare.
- **Invisibilità:** l'invisibilità del software e le difficoltà nel visualizzarlo rendono difficile ragionarci e discuterne.

Per una migliore comprensione del software

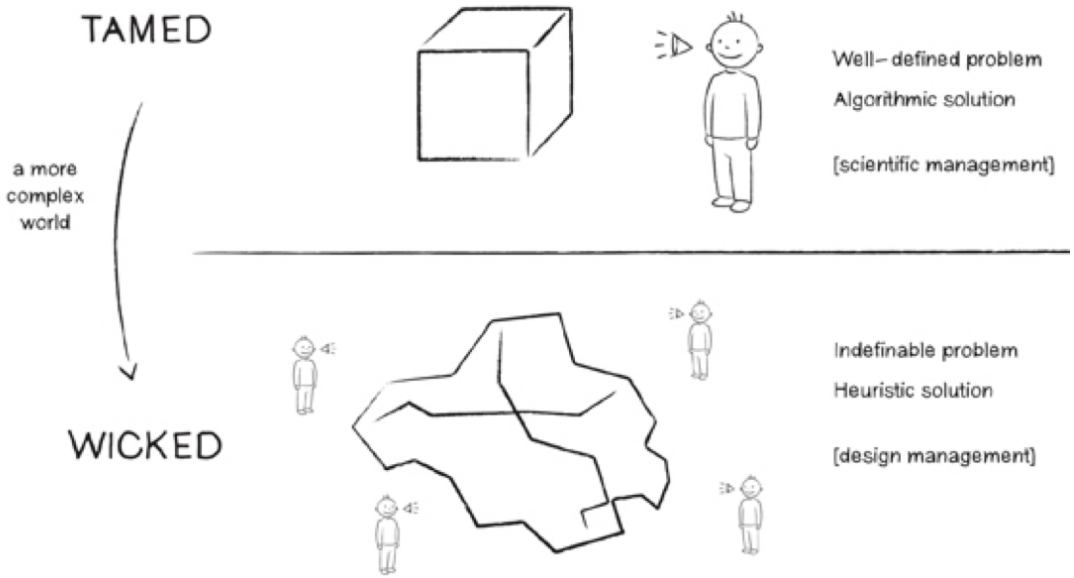
- Se il software non è tangibile, allora potrebbe essere considerato come un'arte, ovvero un atto creativo dell'ingenio
- Il software riguarda la scrittura, quindi potremmo focalizzarci sulla scrittura vera e propria
- Esercizio proposto:
  - leggete e preparate una presentazione di al più 5 minuti con al più 5 lucidi sulle riflessioni che avete elaborato leggendo "Lezioni Americane, Sei proposte per il prossimo millennio" di Italo Calvino qui
  - i gruppi sono liberi
  - utilizzate anche le informazioni che trovate nel resto della lezione
  - la presentazione va fatta usando lo stesso modello del corso utilizzando LaTeX

La comprensione dello sviluppo software

- Per comprendere appieno le problematiche dello sviluppo dei sistemi software, ci concentriamo ora su
  - la natura dei problemi software e la loro non riducibilità a semplici questioni decomponibili
  - i meccanismi di controllo dello sviluppo e del lavoro dei team
  - le modalità con cui diverse persone e diversi team possono lavorare insieme e comporre insieme il prodotto finale

I problemi “tame” e “wicked”

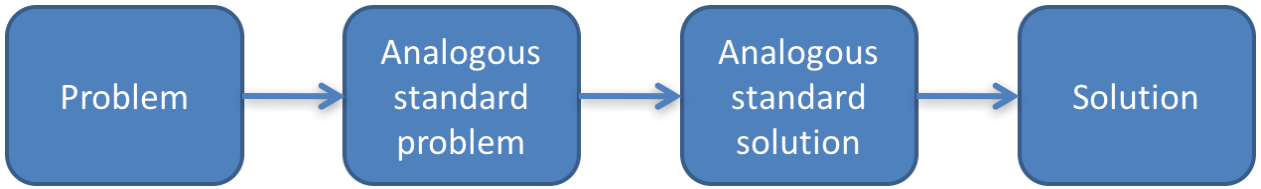
I problemi “tame” sono quei problemi che possono essere ingegnerizzati facilmente; possono essere formulati in modo esaustivo e descritti contentendo tutte le informaizoni necessarie per la loro soluzione.
Non tutti i problemi sono “tame.” ci sono anche i cosiddetti **problemi “wicked”**. Le informaizoni necessarie a descrivere un problema “wicked” dipendono dall’idea che si ha per risolverli.



https://vivifychangecatalyst.wordpress.com/2015/06/28/wicked-problems-innovation-and-project-masiluleke

I problemi tame

- Hanno una descrizione ben definita e stabile
- Hanno un punto di fermata, ovvero una chiara definizione di quando la soluzione è stata trovata
- Hanno una soluzione che può essere definita come giusta o sbagliata
- Appartengono a una classe di problemi simili che possono essere tutti risolti in un modo simile
- Hanno soluzioni che possono essere provate e abbandonate con facilità
- Hanno un insieme limitato di possibili soluzione



http://www.slideshare.net/curtistim/wicked-issues-taming-problems-and-systems

I problemi wicked (1/4)

- I problemi wicked non hanno una formulazione chiara, univoca ed analitica e definire il problema è già un passo verso la soluzione.
- Un problema wicked riguarda spesso una serie di questioni con interdipendenze e vincoli che cambiano nel tempo inseriti in un contesto dinamico e in evoluzione.
- Analogamente, ogni volta si cerca di creare una soluzione si ha una nuova e, si spera, migliore comprensione del problema, e quindi formulare il problema e risolvere il problema sono praticamente la stessa cosa.

I problemi wicked (2/4)

- Il processo di soluzione procede di solito per tentativi successivi finché le risorse per la soluzione sono concluse, gli stakeholder hanno perso interesse in ulteriori raffinamento della soluzione proposta.
- Le soluzioni ai problemi wicked non sono vere o false ma più o meno accettabili. Visto che, come detto, non c'è un criterio non ambiguo per definire il problema come risolto, può essere problematico convincere tutti gli stakeholder che si è raggiunta una soluzione sufficientemente buona.
- Non c'è un test diretto o definitivo per una soluzione a un problema wicked. Soluzioni a questi problemi generano flussi di conseguenze i cui effetti finali sono imprevedibili.

I problemi wicked (3/4)

- Ogni soluzione ad un problema wicked ha conseguenze irreversibili.
  - Bisogna porre attenzione nel gestire presunte soluzioni: ad esempio una volta che una pubblicità di un'offerta è rilasciata è impossibile o, quanto meno, difficoltoso, cambiare tale offerta.
- I problemi wicked non hanno un insieme chiaro e universalmente accettato di possibili soluzioni. I vari stakeholder possono avere opinioni diverse di quelle che sono accettabili.
- Ogni problema wicked è sostanzialmente unico e non è facile trovare problemi analogi precedentemente risolti e ben documentati in modo che le loro soluzioni possano essere replicate.

I problemi wicked (4/4)

- Il problema wicked può essere considerato un sintomo di un altro problema.
- Le cause di un problema wicked possono essere spiegate in modi molto diversi.
- Gli stakeholder hanno spesso idee diverse e pure che cambiano su che cosa sia il problema, la sua natura, le sue cause, e le soluzioni da provare.
- Il problema wicked non può rimanere irrisolto ed il progetto associato non può fallire.
  - Nonostante l'impossibilità di esprimere il problema e la soluzione in modo analitico, il fallimento non è un'opzione accettabile.

Lo sviluppo software è un problema wicked (1/2)

- È molto difficile pianificare all'inizio del progetto tutto lo sviluppo futuro considerando ogni possibile eventualità
- Un prodotto software non è praticamente mai perfetto o finito: quando viene messo in uso subito appaiono nuovo requisiti
- Non esiste una soluzione unica a un problema di ingegneria del software
- Non possiamo sapere a priori il livello a cui una implementazione di un sistema soddisfa i requisiti iniziale finch´è non la implementiamo.
- Le scelte sono alle volte molto costose da essere ribaltate. L'ultima spiaggia è buttare via il prodotto fino a quel punto sviluppato e ricominciare da capo.

Lo sviluppo software è un problema wicked (2/2)

- Ci sono infiniti modi per risolvere un problema di sviluppo software.
- Ogni progetto di sviluppo software è un unicum.
- La presenza di problemi wicked si struttura a livelli multipli: le scelte dei *migliori* database, interfacce utente, sistemi operativi, hardware sono tutti problemi wicked.
- Il problema che lo sviluppo software deve risolvere è visto in modo diverso dai diversi stakeholder del sistema: clienti, utenti, sviluppatori, manutentori, operatori di database, ecc.
- Lo sviluppo software in sé consuma risorse, quindi fare un errore, ovvero sviluppare qualche cosa che il cliente non vuole, non è una opzione accettabile.

Come controllare il processo (1/3)

Michael L. Harris ha proposto un meccanismo efficace per classificare i diversi tipi di controllo e le circostanze in cui funzionano meglio.

La selezione del meccanismo più efficace dipende da due fattori:

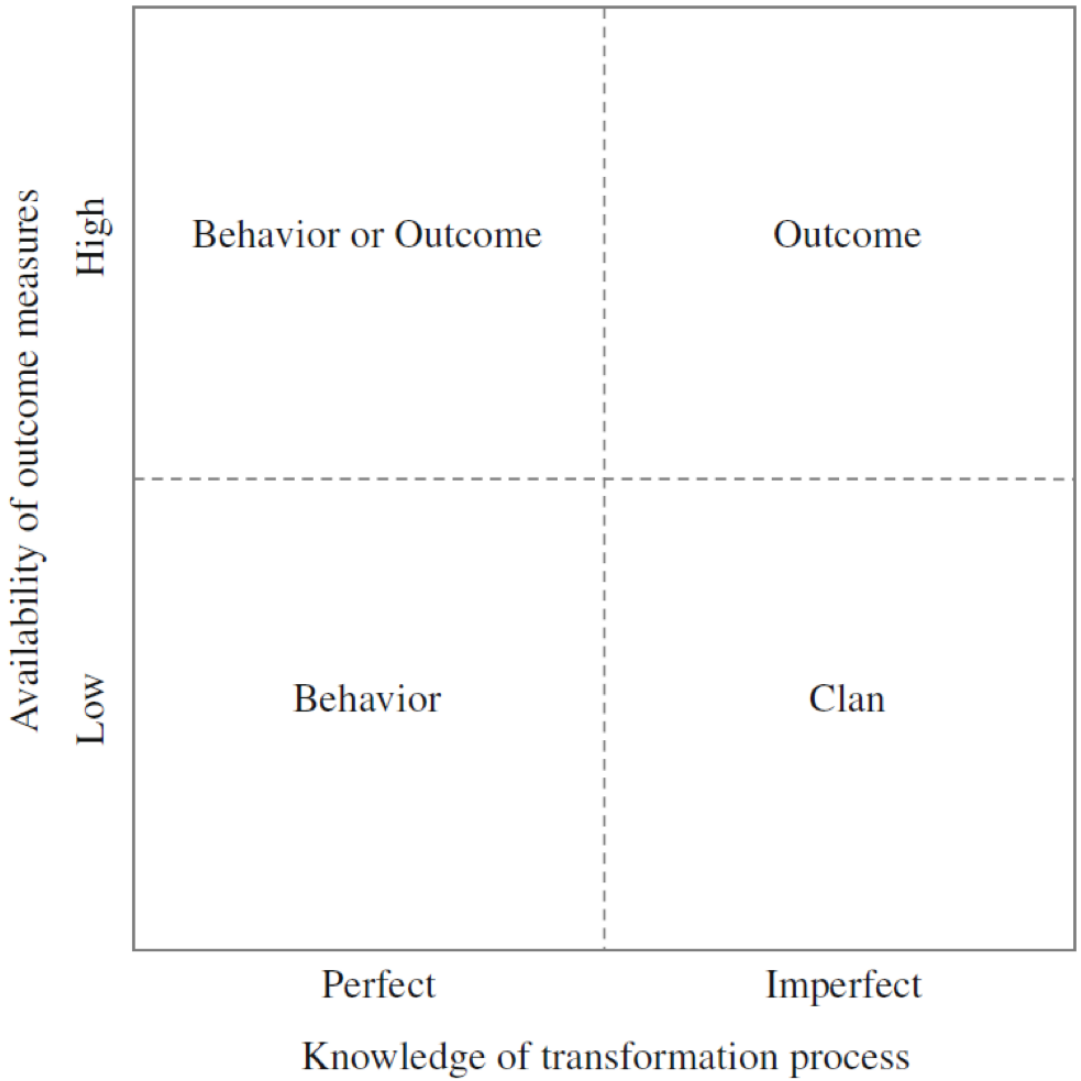
- l'abilità di misurare il risultato (output), ovvero la "misurabilità" e
- l'abilità di specificare in dettaglio i passi necessari per svolgere un dato compito, la "specificabilità".

Come controllare il processo (2/3)

Mettendo questi due fattori in un diagramma bidimensionale otteniamo quattro aree::

- l'area con il più alto livello di misurabilità e specificabilità;
- l'area in cui si ha un alto livello di misurabilità ma poca specificabilità;
- l'area con poca misurabilità e con il più alto livello di specificabilità;
- l'area in cui non possiamo né misurare con precisione il risultato né definire con precisione i risultati del problema

Come controllare il processo (3/3)



Meccanismi di coordinamento

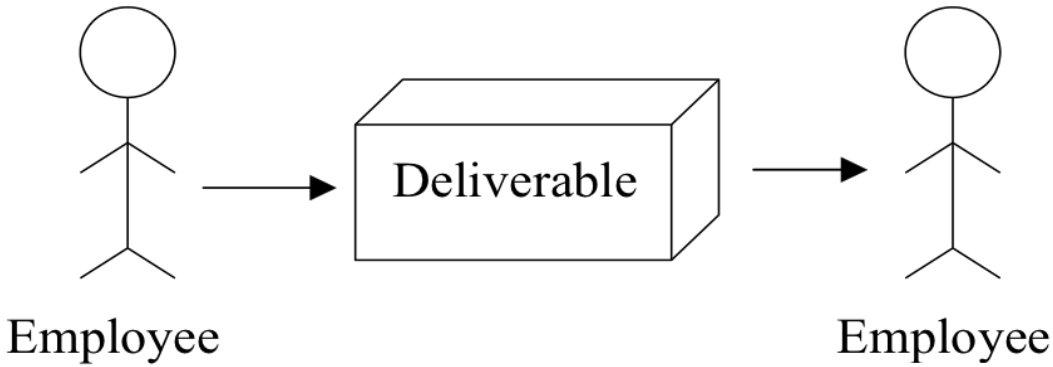
7

Secondo Malone e Crowston (1994) ci sono tre modi con cui le persone si coordinano:

- sequenziale o individuale (sequential)
- tramite risorse condivise (shared resources)
- tramite risultato comune (common output)

Coordinamento sequenziale

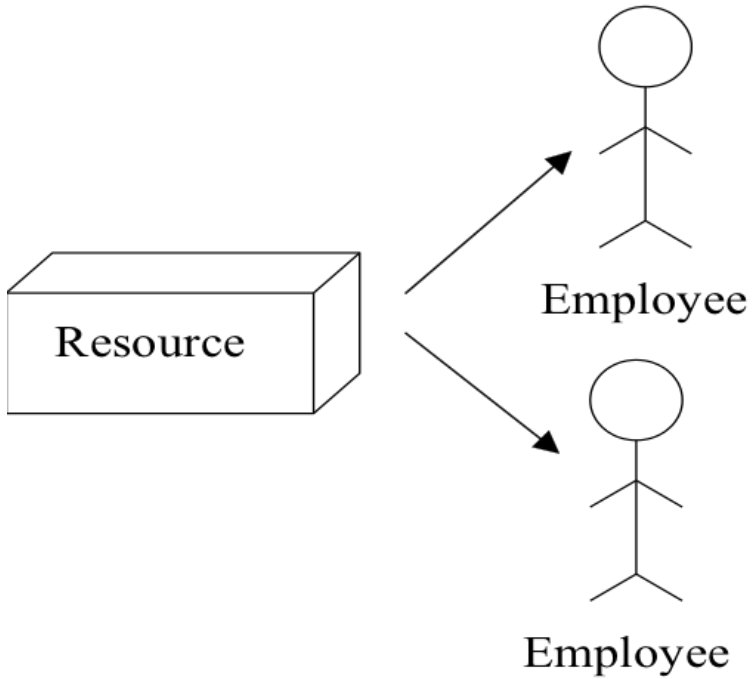
8



- Prima si scrive il documento di analisi
- Sulla base del documento di analisi si progetta il sistema

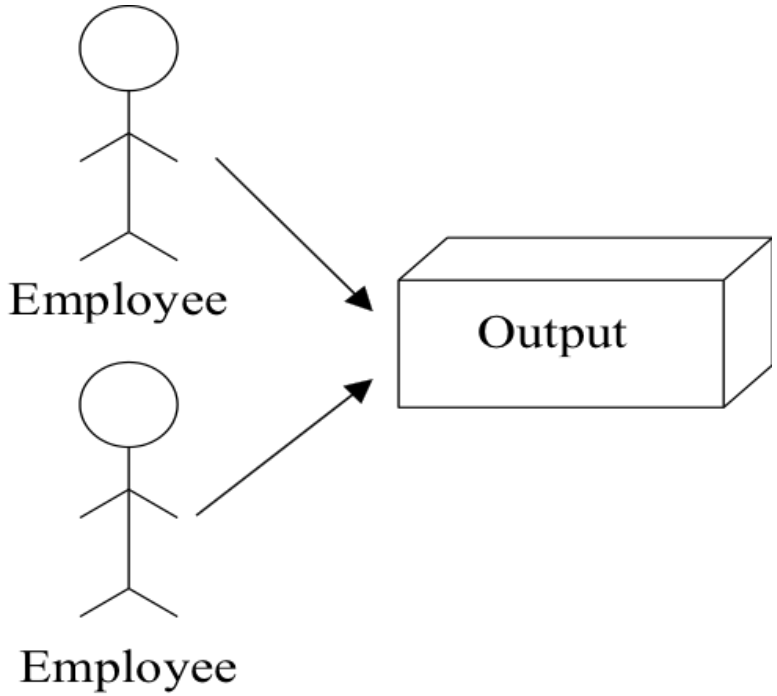
Coordinamento a risorse condivise

9



- Lo sviluppo può iniziare quanto il cliente ha dato il via al progetto
- Le parti da sviluppare possono essere assegnate agli sviluppatori con un meccanismo a coda (first come, first served)

Coordinamento a risultato comune



- Il responsabile del team propone un piano di lavoro per la settimane e tale piano viene approvato se il team unanimamente lo accetta
- Il team si siede insieme e produce la versione finale che richiede l'approvazione di tutti

Qual è il più difficile?

1

- Il più facile: quello sequenziale!
- Poi: risorse condivise – occorre saper rispettare le priorità di sviluppo
- Il più difficile: risultato condiviso
  - le persone devono lavorare insieme, mano nella mano, producendo lo stesso bene
- La gran parte dei modelli tradizionali di sviluppo software si basano su coordinamento sequenziale

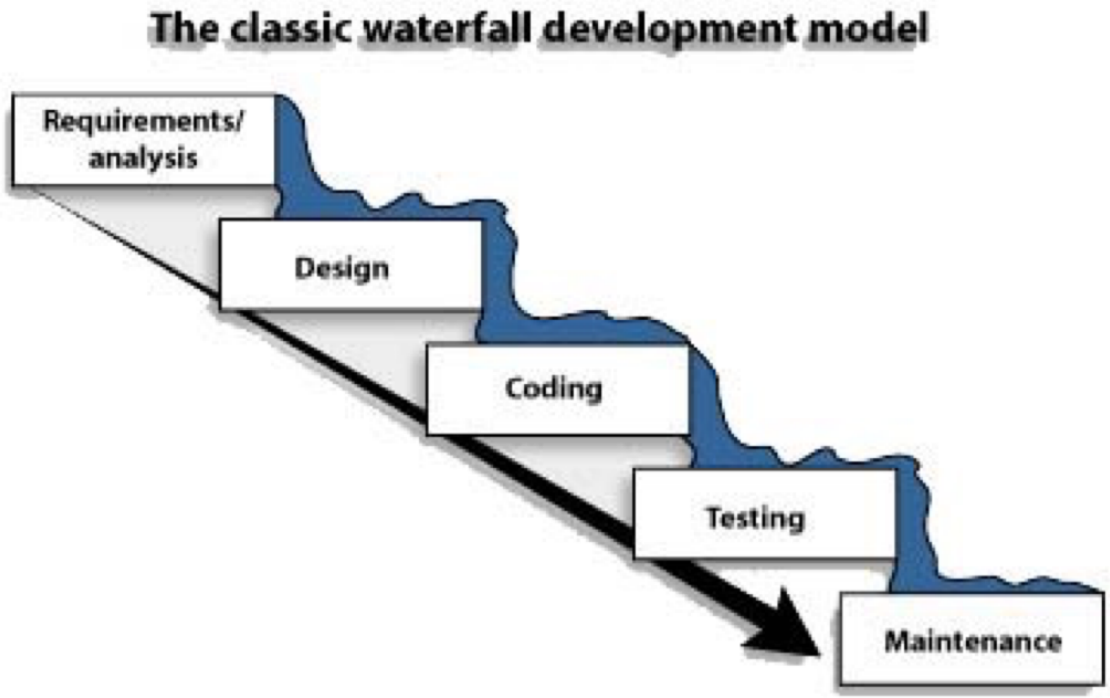
I modelli di sviluppo software

- Ora analizziamo brevemente i modelli di sviluppo software che sono stati proposti negli anni
- Cerchiamo di comprendere i meccanismi di controllo e coordinamento in essere per determinare:
  - la loro efficacia
  - la loro difficoltà di realizzazione

Il modello di sviluppo a cascata

- Il modello di sviluppo a cascata è stato il primo modo con cui si è tentato di riuscire a creare un prodotto software che risponda ai requisiti.
- Per ottenere tale risultato, si sono definiti passi precisi che rispecchiano il mondo manufatturiero in cui, ad esempio, la progettazione avviene dopo l'analisi, lo sviluppo dopo la pregettazione, e la verifica di correttezza dopo lo sviluppo.
- Il tutto sembra molto logico ...

Il modello di sviluppo a cascata – schema



Fonte: http://www.railshorde.com/blog/waterfall-development-model

Il modello di sviluppo a cascata – fasi (1/3)

Pianificazione e Requisiti: Tutti i possibili requisiti del sistema da sviluppare sono raccolti e viene sviluppato un piano per lo sviluppo affinché tutto sia fatto nei tempi e nei costi prestabiliti.

Analisi e Progettazione del sistema: I requisiti precedentemente elicitati sono messi in forma scritta, analizzati con cura e quindi viene sviluppato un progetto (design) del sistema. Questo progetto rappresenta il riferimento del software da sviluppare. Il progetto di sistema serve a definire i bisogni di hardware e di software e specifica anche l'architettura di massima.

Fonte: http://www.railshorde.com/blog/waterfall-development-model

Il modello di sviluppo a cascata – fasi (2/3)

Costruizione: Con le informazioni raccolte nei documenti di progetto incomincia la codifica. Gli sviluppatori sono responsabili per il codice prodotto. Di solti si parte focalizzandosi su piccole unità e poi il sistema cresce componendo tali piccole unità in entità più "grandi" per la successiva integrazione. Si noti che il concetto di grande è molto soggettivo.

Test di unità, integrazione e test finale di sistema: Tutte le unità sviluppate singolarmente nella fase di costruzione sono verificate (testate) singolarmente, e quindi integrate nel prodotto finale. Il prodotto finale completo è quindi testato complessivamente per rilevare ulteriori errori, difetti, non conformità con i desideri dei clienti.

Fonte: http://www.railshorde.com/blog/waterfall-development-model

Il modello di sviluppo a cascata – fasi (3/3)

Consegna del sistema: Dopo che ci si è assicurati della qualità del prodotto con il test, il prodotto viene consegnato al cliente.

Manutenzione: Per gestire tutte le problematiche che avvengono dopo la consegna del prodotto, una specifica fase di manutenzione viene prevista. Inoltre, data la costante evoluzione dei sistemi hardware e software la fase di manutenzione si occupa dell'aggiornamento del sistema in modo che il cliente abbia sempre un sistema aggiornato ed efficace.

Fonte: http://www.railshorde.com/blog/waterfall-development-model

Il modello di sviluppo a cascata – commento

Ovviamente tutto quanto detto sopra è lo sviluppo a cascata nei desideri di chi lo ha ideato ... una sorta di lettera a babbo natale!

Fonte: https://www.triesteallnews.it/wp-content/images/2019/12/san_nicolo-201x300.jpg

Vantaggi del modello di sviluppo a cascata

- Semplice da usare e capire.
- Ogni fase viene sviluppata individualmente in modo che ci si dedica a fondo e con concentrazione ad una azione alla volta.
- Funziona bene per piccoli progetti dove i requisiti sono ben compresi.
- Ogni fase ha le attività che deve svolgere in modo preciso.
- Ogni fase è documentata nel dettaglio per evitare qualunque possibile ambiguità.
- Notate di nuovo che questi sono i desiderata . . . che poi la realtà corrisponda ad essi, è un altro discorso.

Svantaggi del modello di sviluppo a cascata

- Il primo svantaggio che appare è che il tutto è molto sequenziale e si può fare solo una cosa alla volta; questo confligge con la natura combinatoria e iterativa dei problemi wicked.
- I requisiti devono essere definiti all'inizio; se un requisito viene identificato dopo, esso non può essere inserito con facilità nel processo di sviluppo.
- Il modello a cascata non permette facilmente iterazioni e il progetto nella sua interezza va integrato alla fine.
- I clienti possono avere accesso al progetto solo alla fine. Non sono rilasciati prototipi o sistemi parziali in anticipo.

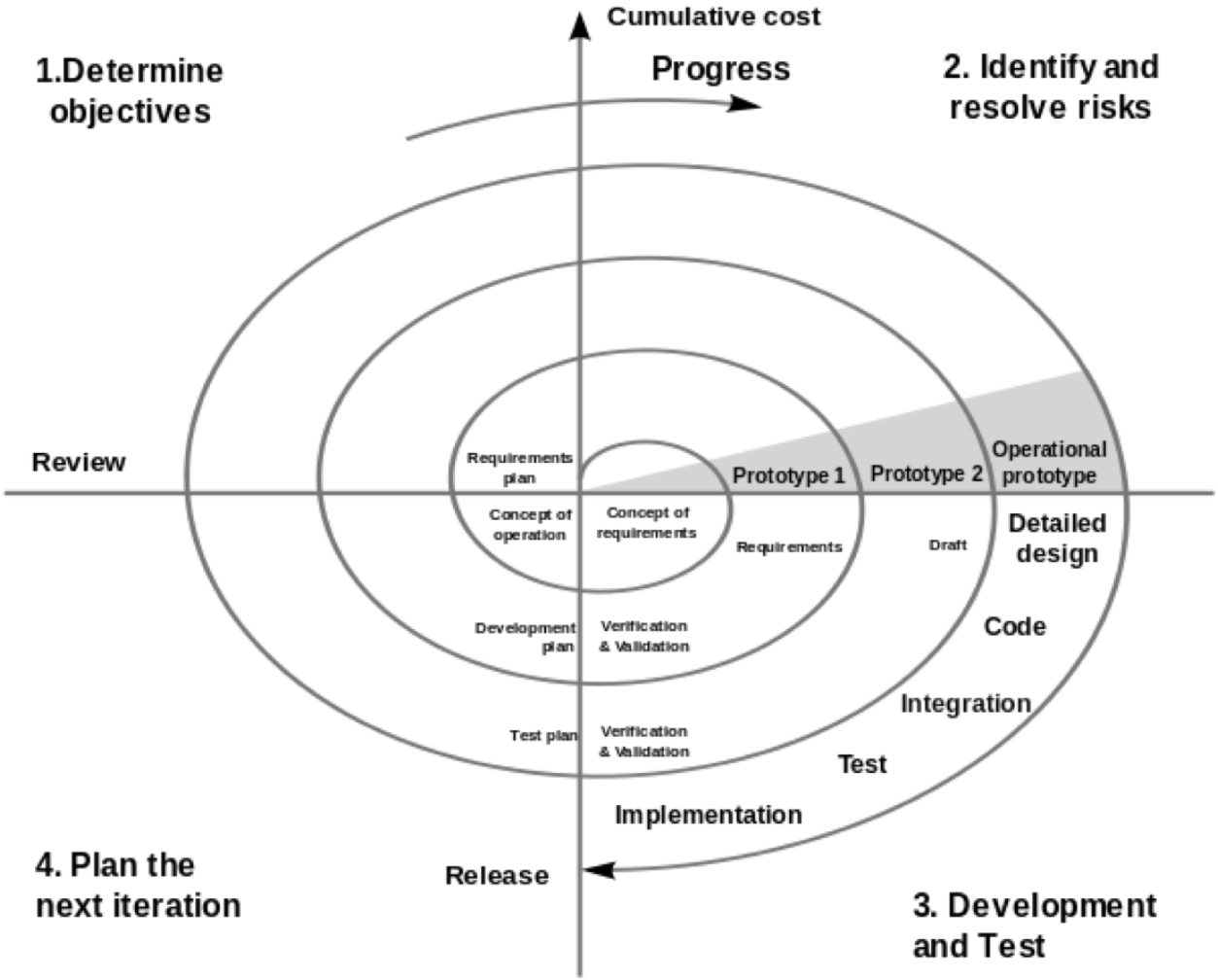
Applicabilità del modello di sviluppo a cascata

- Non ci devono essere requisiti ambigui.
- I requisiti devono essere ben documentati e fissati in anticipo.
- La tecnologia in uso (hardware, sistema operativo, API, ...) non deve cambiare durante il progetto.
- La definizione del prodotto deve essere stabile.
- Risorse con esperienza devono essere presenti per aiutare lo sviluppo del prodotto.

Evoluzioni e raffinamenti

- Il modello di sviluppo a V, con gli associati
  - modello a dente di sega e
  - modello a dente di squalo
- Lo sviluppo incrementale.
- Il modello prototipale.

Modello di sviluppo a spirale



Fonte: https://en.wikipedia.org/wiki/Spiral_model

Principi del modello di sviluppo a spirale

Il modello di sviluppo a spirale è un approccio iterativo allo sviluppo software in cui ogni ciclo della spirale ha le seguenti caratteristiche:

- Determinazione degli artifatti da sviluppare concorrente e non sequenziale
- Inclusione in ogni ciclo della spirale dei sequenti elementi:
  - revisione degli obiettivi e dei vincoli degli stakeholder critici del sistema,
  - analisi di processi e prodotti alternativi,
  - identificazione e risoluzione dei rischi,
  - revisione con gli stakeholder,
  - esplicita decisione di continuare o terminazione del progetto.

Ingredienti del modello di sviluppo a spirale

- Uso dell'analisi del rischio per determinare il livello di sforzo necessario ad ogni attività all'interno di ogni ciclo della spirale
- Gestione all'interno di ogni ciclo della spirale degli accordi con gli stakeholder sulla struttura e sul contenuto del processo di sviluppo.
- Enfasi su attività e artefatti del sistema e del processo di produzione nel loro insieme e non soltanto sul software risultante alla fine.

L'Agile Manifesto (1/2)

- Nel Febbraio 2001 un gruppo di softwaristi molto esperti formalizzò un nuovo modo di sviluppare software in un Manifesto
- Questo manifesto sintetizzava pratiche che si erano sviluppate nel tempo, alcune che risalivano agli albori dello svilupo software
- Di fatto il manifesto era una istanziazione del modello di sviluppo a spirale
- Al manifesto faceva poi seguito una lista di principi

L'Agile Manifesto(2/2)

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.

Fonte: https://agilemanifesto.org

Agile Manifesto

L'"Agile Manifesto" identifica due insiemi di valori:

- quelli a destra del documento
- quelli a sinistra del documento

Values on the left	Values on the right
Individual and interactions	Processes and tools
Working software	Comprehensive documentation
Customer collaboration	Contract negotiation
Responding to change	Following a plan

Principi dell'Agile Manifesto (1/2)

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software
- Welcome changing requirements, even late in development, Agile processes harness change for the customer's advantage
- Business people and developers must work together daily throughout the project
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale
- Build projects around motivated individuals
- Give them the environment and support they need, and trust them to get the job done
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation

Fonte: https://http://agilemanifesto.org/principles.html

Principi dell'Agile Manifesto (2/2)

- Working software is the primary measure of progress
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely
- Continuous attention to technical excellence and good design enhances agility
- Simplicity – the art of maximizing the amount of work not done – is essential
- The best architectures, requirements, and designs emerge from self-organizing teams
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly

Fonte: https://http://agilemanifesto.org/principles.html

Domande?

Fine della lezione cinque.