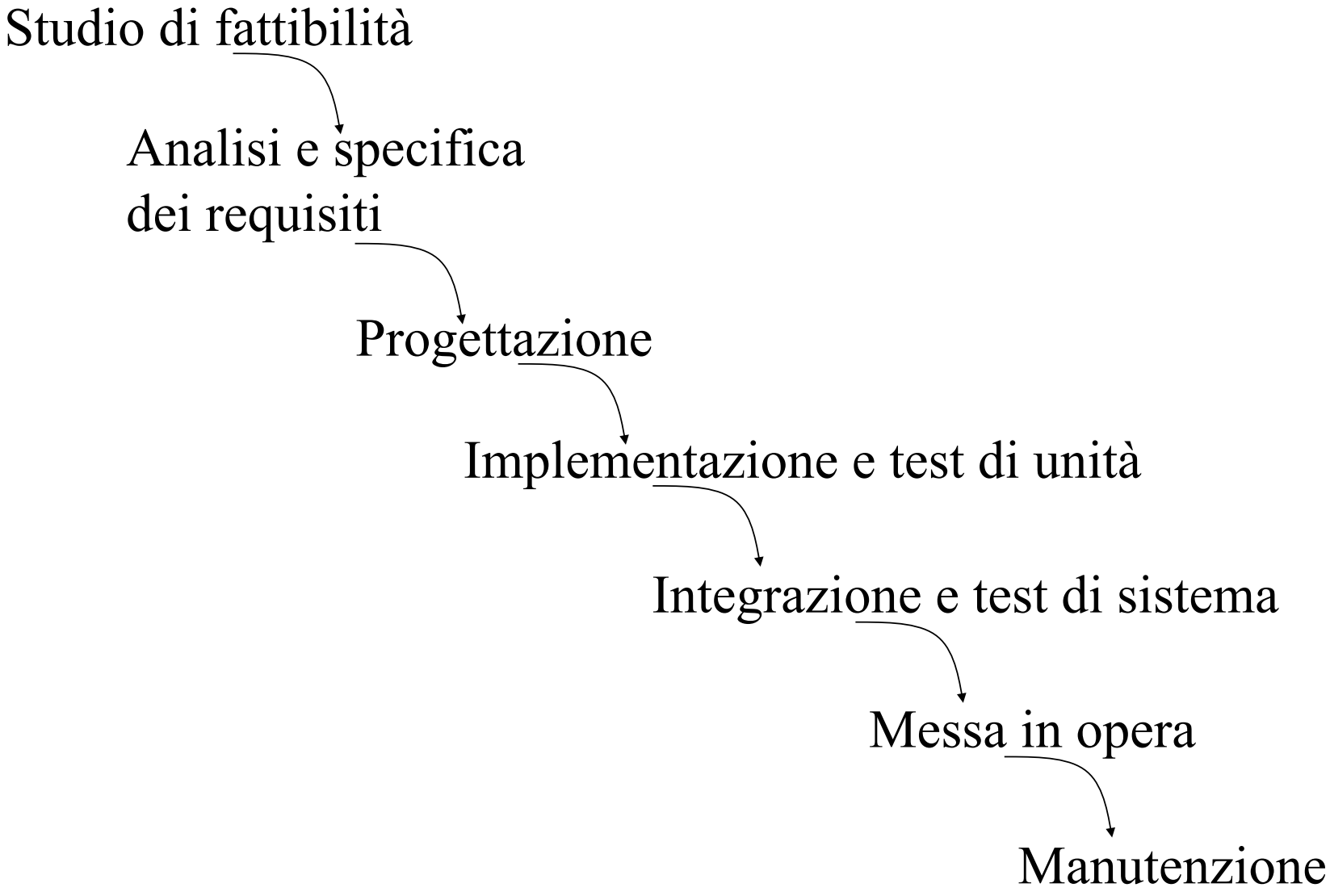
Il processo di sviluppo del software

Dal nulla al modello “a cascata”

- All’inizio (e talvolta anche ora) si procedeva senza un modello di riferimento
  - “Code & fix”
- Con la nascita dell’ingegneria del software, negli anni 70 si è proposto (e tuttora si usa) il ciclo di vita a cascata (**waterfall**)
  - Identifica fasi e attività predefinite
  - Forza una progressione lineare tra una fase e la successiva attraverso decisioni **go/no-go**
  - Niente ritorni all’indietro (considerati pericolosi perchè stravolgono le stime e lo sviluppo ordinato)

Modello a cascata (Royce, 1970)



Ricorda qualcosa?

Studio di fattibilità

- Obiettivo: decisione go/no-go
  - Si deve fare il progetto?
  - Buy or make?
  - Analisi costi/benefici
- Alternative possibili, costi associati e risorse necessarie
- Produce il documento "**Studio di Fattibilità**"
  - Descrizione preliminare del problema
  - Scenari che descrivono le possibili soluzioni
  - Costi e tempi previsti per ogni alternativa

Analisi e specifica dei requisiti

- Analizza il dominio in cui l'applicazione si colloca
- Identifica i requisiti
- Deriva una specifica per il software
  - Necessaria una comprensione del dominio
  - Necessaria l'interazione con gli "stakeholders"
- Produce il documento "**Analisi e Specifica dei Requisiti**"

Progettazione

- Definisce l'architettura del software
  - Componenti (moduli)
    - Come vedremo, da "sottosistemi" a singoli "sottoprogrammi"
  - Relazioni tra componenti
- Goal: supportare lo sviluppo concorrente, definendo le diverse responsabilità per le diverse parti
- Produce il documento "**Progetto del Sistema**"

Implementazione e test di unità

- Ogni modulo elementare viene implementato mediante il linguaggio di programmazione prescelto
- Ogni modulo viene testato dallo sviluppatore mentre viene sviluppato
- I programmi includono la loro documentazione

Integrazione e test di sistema

- I moduli sono integrati in (sotto)sistemi e ciascun sottosistema integrato viene testato
- La fase può essere integrata con la precedente in uno schema di implementazione incrementale
- Il sistema completo viene testato alla fine per verificare proprietà complessive (es. tempi di risposta) che non avrebbero senso sui singoli componenti
- Talvolta si effettuano "**alpha test**" (in "casa" dello sviluppatore) e "**beta test**" (presso utenti "selezionati")

Distribuzione degli sforzi

- Regola "iper-grossolana"
  - 40% requisiti e progettazione
  - 30% implementazione
  - 30% testing
- Variazioni molto forti possibili

Deployment e manutenzione

- Il deployment ha l'obiettivo di distribuire l'applicazione e gestire le diverse installazioni e configurazioni presso i clienti
- La manutenzione copre i cambiamenti successivi alla fine del proceso di sviluppo ("post-delivery")
- Termine infelice: il software non si deteriora
  - Se sorge un malfunzionamento, la causa dello stesso è presente fin dall'inizio
- Supera il 50% dei costi complessivi
  - Survey di imprese Europee: 80% del budget IT

Correzione o evoluzione?

- Correzione necessaria a fronte di un'implementazione che non rispetta la specifica
- La specifica potrebbe far riferimento a requisiti "sbagliati", di cui il committente non si è accorto
- Ciò diventa particolarmente problematico se la specifica dei requisiti è fatta male (incompleta, imprecisa)

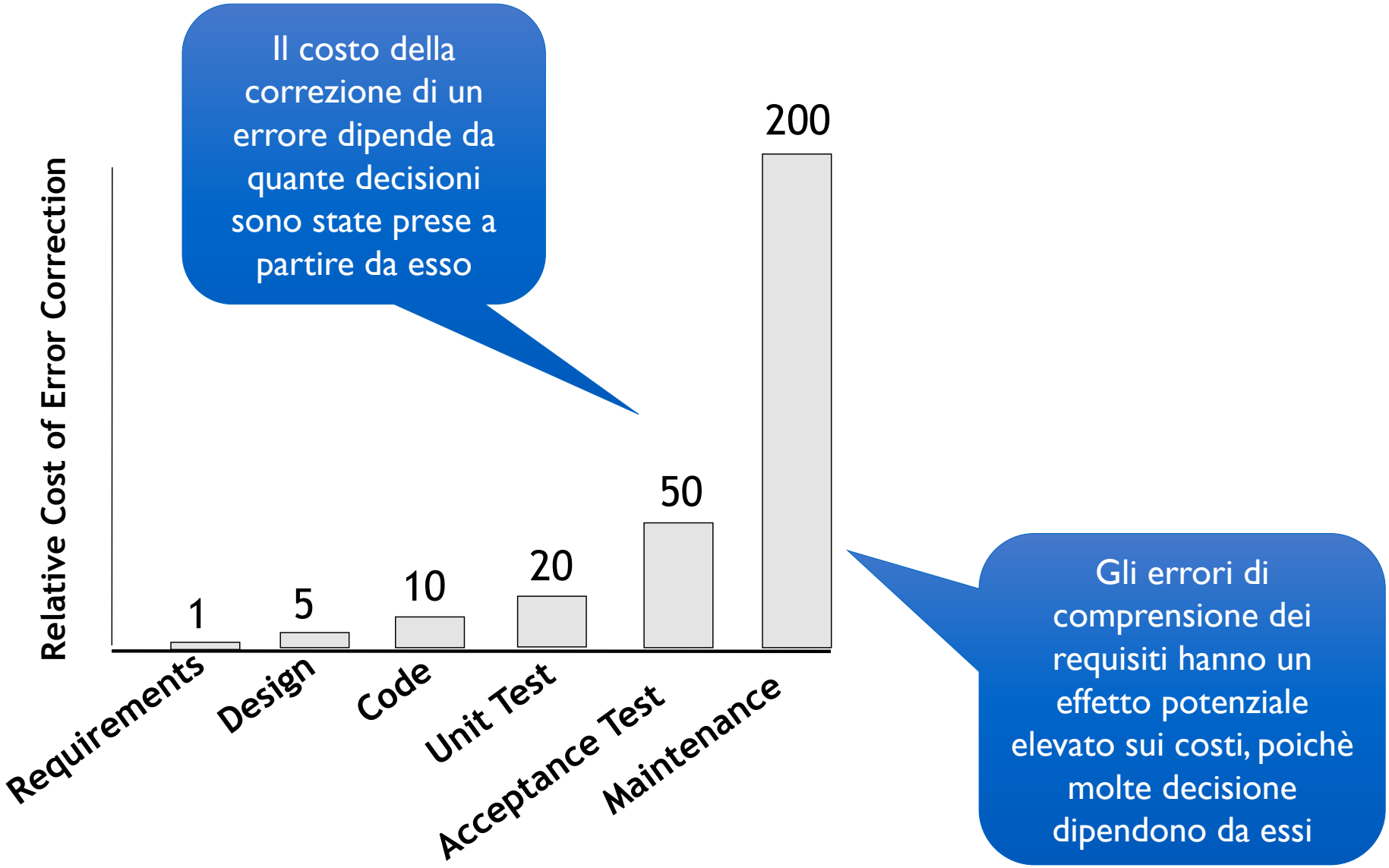
Perchè il software evolve?

- Cambiamenti nel contesto
  - Introduzione dell'EURO
  - Cambiamenti normativi
- Cambiamenti nei requisiti
  - L'introduzione di una nuova versione del sistema genera nuova domanda
    - Secondo un survey delle aziende Europee, il 20% dei requisiti utente è obsoleto dopo 1 anno
- Requisiti non noti inizialmente

Verifica e convalida

- Talvolta si distingue tra l'attività di **verifica** (nel nostro caso, via testing) con la quale si effettua un controllo che il software rispetti le specifiche
- Rispetto alla **convalida** con la quale si verifica che il software soddisfi le aspettative del committente
- Se le specifiche catturano le aspettative del committente, non esiste differenza tra i due concetti!

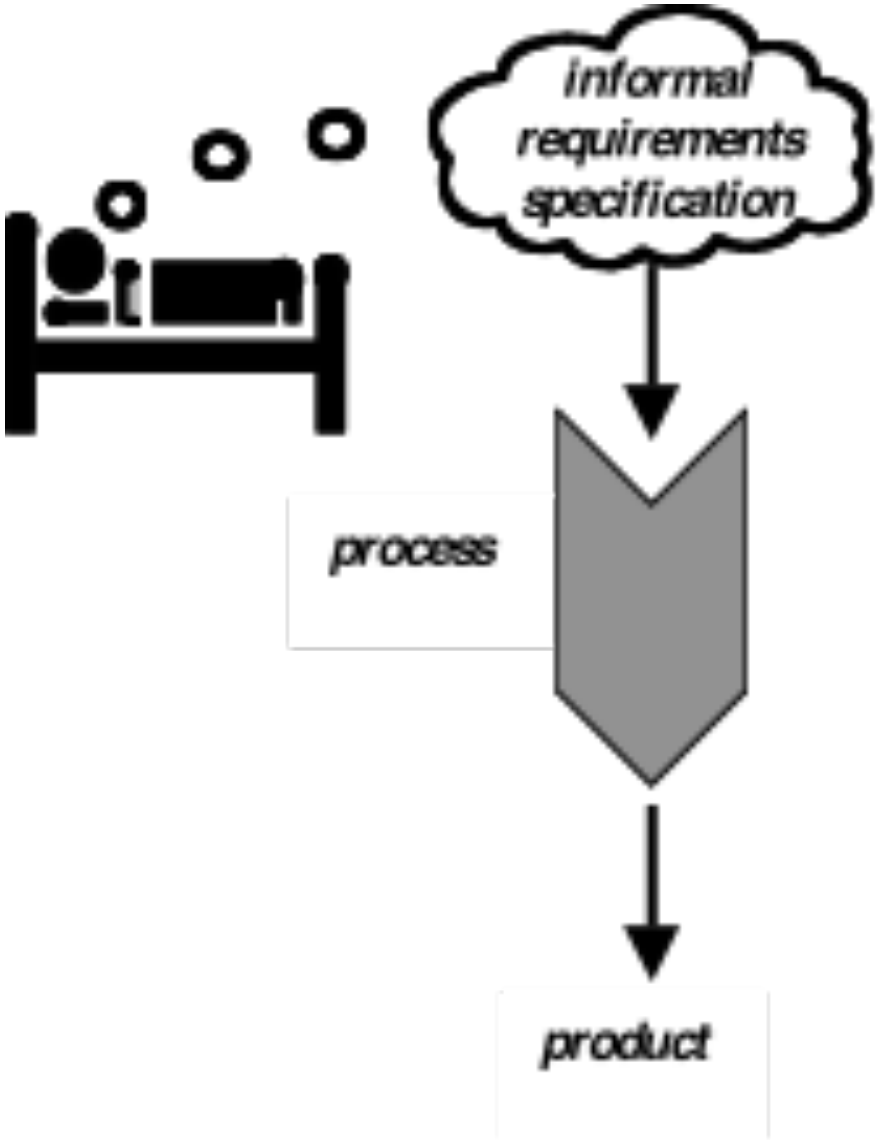
Costo delle correzioni ritardate (Boehm, 1981)



Come fronteggiare il cambiamento?

- Anticiparlo, non subirlo
- Il ciclo di vita a cascata non offre certamente un buon punto di partenza!
- Vedremo qualche alternativa
- Vedremo come progettare il software per favorirne il cambiamento in maniera economica e affidabile

Il ciclo a cascata è "black box"



Problemi del ciclo a cascata

- Adatto alle situazioni in cui il dominio è perfettamente noto e i requisiti sono noti e stabili
- Ciò accade **raramente**
- In pratica sono spesso necessarie iterazioni successive per arrivare progressivamente al prodotto desiderato

Cicli di vita “agili”

- Vedi http://agilemanifesto.org/
- Si adattano ai cambiamenti
- Procedono in maniera incrementale, per successivi rilasci
- Esistono in varie forme
- Le più note
  - SCRUM
  - Extreme Programming (XP)

SCRUM

- Metodologia iterativa e incrementale
- Parte dal riconoscimento che i committenti possono cambiare idea e che il processo a cascata non è in grado di reggere a questa "turbolenza"
- Nata in Giappone al di fuori dell'industria del software
- Nome basato su analogia rugbystica relativa a un attacco coordinato all'avversario basato su avanzamento per fasi

Caratteristiche salienti (I)

- Ruoli
  - Scrum Master
    - Responsabile del processo
  - Product Owner
    - Rappresenta gli stakeholders
  - Team
    - Gruppo di circa 7 persone responsabili dell'analisi, progetto, implementazione, testing, ...

Caratteristiche salienti (II)

- Organizzazione del progetto in **sprint**
  - Periodo da 4 a 7 settimane (durata decisa dal team) nella quale si crea un incremento di prodotto rilasciabile
- **Funzionalità** dello sprint
  - Scelte da un product **backlog** del progetto nel quale le funzionalità sono marcate da una priorità dei relativi requisiti che esse devono soddisfare
- Durante lo sprint il backlog **non** può essere cambiato
  - Requisiti congelati durante lo sprint
- Al termine dello sprint (durata predefinita!)
  - I requisiti non soddisfatti tornano nel backlog

Extreme programming (XP)

- Si basa sull'analogia dell'extreme climbing
- Ha per obiettivo la risposta a requisiti stringenti di ridotto "time to market"
- Si basa su una premessa cha ha un senso ma che non può essere generalizzata
  - Il sw è **impossible** da prespecificare
  - … è come guidare: anche se si ha in mente un piano, occorre un continuo riaggiustamento

Caratteristiche fondamentali

- Come SCRUM rivendica la centralità del codice rispetto ad altri artefatti
  - Con il conseguente pericolo di ricadere nel code-and-fix…
- Integra il testing nel processo come attività centrale
  - **Test first**: prima di iniziare a codificare, si definiscono i casi di test che dovranno essere superati
- **Pair** programming
  - …ogni lavoro di implementazione è fatto da due programmatori appaiati
- Refactoring continuo

Prototipazione rapida

- Talvolta nel processo di sviluppo del software si possono realizzare prototipi
  - Per meglio capire ciò che si dovrà poi realizzare
  - Ciò è relativamente comune in altri processi produttivi
  - Il prototipo può differire dal prodotto per diverse caratteristiche
- Nel caso del software il prototipo può essere
  - **Throw-away**
  - **Evolutivo**

Conclusioni

- Esistono diversi tipi di cicli di vita
  - Abbiamo visto i due estremi (a cascata, XP)
- Basati su diverse tipologie di prodotti e per diverse organizzazioni
  - Per esempio, secondo uno dei proponenti di XP

XP is a lightweight methodology for small-to-medium-sized teams developing software in the face of vague or rapidly changing requirements (K. Beck, 2000)

  - Applicazioni critiche richiedono un approccio molto più strutturato
- In generale, il ciclo di vita va **progettato in funzione del caso specifico**

Processo e prodotto

Processo e prodotto

- Il nostro obiettivo è produrre prodotti di qualità
- Il processo (vedi anche ciclo di vita) è come ciò avviene
- Entrambi sono importanti
- Entrambi hanno **qualità**

Qualità di prodotto: correttezza

- Un software è corretto se soddisfa le sue specifiche
- Se le specifiche sono espresse in maniera formale, la verifica della correttezza può essere definita formalmente (matematicamente)
  - Può essere dimostrata come teorema o falsificata da controesempi attraverso il testing
- Definita in maniera assoluta (SI/NO)
  - Non esiste, anche se sarebbe utile, un concetto di "grado di correttezza"

Affidabilità, robustezza

- Affidabilità
  - Informalmente, "l'utente si può fidare"
  - Definibile matematicamente come "probabilità dell'assenza di un malfunzionamento per un certo periodo di tempo"
- Robustezza
  - Il software si comporta "ragionevolmente" anche in circostanze inattese (e.g., input scorretti, malfunzionamenti hardware)

Prestazioni

- Uso efficiente delle risorse
  - …memoria, processori, banda di rete, …
- Può essere verificata mediante
    - Analisi di complessità
    - Performance evaluation (su modelli, via simulazione)
- Le prestazioni hanno effetto sulla "scalabilità"
    - La soluzione funziona anche a scale diverse di alcune caratteristiche
      - …piccola rete locale rispetto a Internet
- Le prestazioni possono aver effetto sull'usabilità

Usabilità

- Gli utenti attesi trovano il software facile da usare
- Importante definire gli utenti attesi
  - …per l'utente installatore la procedura di installazione deve essere di uso facile
- Altri termini: ergonomico, user friendly
  - Largamente dovuta all'interfaccia utente
    - …testuale vs grafica
- Qualità largamente soggettiva e difficile da valutare

Altre qualità

- Manutenibilità
- Riusabilità
  - Simile alla precedente, ma si applica a "componenti"
- Portabilità
  - Adattamento a diversi ambienti target
- Interoperabilità
  - Coesiste e coopera con altre applicazioni

Qualità di processo: produttività

- Produttività
  - …come si definisce e misura?
    - Quantità prodotta per unità di lavoro (effort)
- Unità di lavoro: mese persona
    - **Attenzione**: persone e mesi non sono interscambiabili
- Quantità prodotta
  - Linee di codice (e variazioni)
  - Punti funzione
- Alcuni di voi lo vedranno ad Ingegneria del Software II…