INGEGNERIA DEL SOFTWARE

- software engineering -

A.A 2016/2017

Nota introduttiva:
I seguenti appunti contengono un riassunto/glossario relativamente alla parte teorica del corso “Ingegneria del Software”, tenuto dal professor Tullio Vardanega (A.A 2016/2017).
Non garantisco la completezza/correttezza di tali appunti; vi invito pertanto a leggere e valutare criticamente quanto scritto qui di seguito.
Alla fine del documento è inoltre presente un glossario, contenente termini che non ho ritenuto essere argomenti centrali delle lezioni ma direttamente collegabili ad alcune di esse. Nel documento, le parole presenti nel glossario sono riconoscibili dalla dicitura “(← glossario)”.
Gli appunti contengono, tra l'altro, integrazioni basate su appunti attualmente presenti su MEGA; voglio quindi ringraziare gli autori di tali appunti, che hanno deciso di condividere il loro lavoro.
Buona lettura.

Daniel De Gaspari

Lezione T1 – Data: 03/10/2016 – Titolo: **Introduzione**

Introduzione all'ingegneria del software:

Cos'è l'ingegneria del software?

- **ENGINEERING:**
  - Application of scientific and mathematical principles to practical ends
  - Questi “practical ends” (= frutto dell’attività di engineering) sono civili e sociali e non solo prodotti di consumo (infrastrutture, servizi)
    - ciò comporta dunque responsabilità etiche e professionali
  - Questa disciplina ha dunque come obiettivo l’applicazione di conoscenze e risultati delle scienze matematiche, fisiche e naturali volte alla risoluzione di problemi riguardanti la soddisfazione dei bisogni umani materiali nella società, attraverso lo sviluppo di sistemi in grado di soddisfare tali bisogni.

- **INGEGNERIA DEL SOFTWARE:**
  - Cosa succede quando questa definizione [engineering] si applica al software?

  - **Definizione secondo IEEE:**
    - Approccio **sistematico, disciplinato** e **quantificabile** allo **sviluppo**, **l’uso**, la **manutenzione** e il **ritiro** del SW
      - **Approccio Sistematico** = **metodico:**
        - rispettare un metodo noto (=non inventato)
          - una check-list, secondo un certo ordine
        - presa un’altra persona e messa a fare la stessa cosa, la farà con la stessa sequenza di azioni, con lo stesso approccio
      - **Approccio Disciplinato:**
        - insieme di norme che seguo, o meglio ancora delle **best practice** (← glossario)
      - **Approccio Quantificabile:**
        - so quanto consumerò, con una stima ragionevole

  - Disciplina che si occupa dei processi produttivi e delle metodologie di sviluppo finalizzate alla realizzazione di sistemi software. Lo scopo è quello di realizzare sistemi SW così impegnativi da richiedere lavoro di gruppo
    - capacità di produrre “in grande” e “in piccolo”
    - garantire la qualità dei prodotti (=**Efficacia**) (← glossario)
    - contenere i costi e i tempi di produzione (=**Efficienza**) (← glossario)
      - lungo l’ intero **ciclo di vita** del prodotto

- Si tratta quindi di applicare i principi ingegneristici al SW
  - Il SW è un prodotto con un proprio **ciclo di vita**
    - la sua manutenzione costa molto spesso più della sua produzione
    - i costi di produzione sono spesso dominati dai costi di verifica

- Il controllo di **efficienza** ed **efficacia** ha bisogno di un approccio sistematico

- Il frutto dell'attività del software engineering:
  - un servizio
  - un'applicazione
  - es: Uniweb fornisce vari servizi

- Relazione con le altre discipline:
  - *The scientific disciplines of reference to SWE include not only computer science, but also certain areas of discrete mathematics and operation research, statistics, psycohology and economics*
  - ***SWE isn't a branch of computer science**; it is an engineering discipline relying in part on computer science, in the same way that mechanical engineering relies on pysics*
  - L'ambito ingegneristico applica, non inventa principi matematici

- Prodotto Software:
  - Esistono svariate **tipologie di prodotto SW:**
    - **Su commessa:**
      - forma, contenuto e funzione fissate dal cliente
    - **Pacchetto:**
      - forma, contenuto e funzione idonee alla replicazione
    - **Componente:**
      - forma, contenuto e funzione adatte alla composizione
    - **Servizio:**
      - forma, contenuto e funzione fissate dal problema

- **Forme di manutenzione:**
  - Durante il proprio **ciclo di vita**, molti sistemi vengono sottoposti a svariate forme di manutenzione:
    - **Correttiva:**
      - **Bug fixing**
        - per correggere difetti eventualmente rilevati
        - la peggiore forma di manutenzione agli occhi degli utenti, in quanto segnala software scritto male.
    - **Adattiva:**
      - per adattare il sistema alla **variazione dei requisiti** (← glossario)
      - tiene inoltre conto dei feedback degli utenti
    - **Evolutiva:**
      - per **aggiungere funzionalità al sistema**
      - il prodotto subisce un'evoluzione a fronte di nuove tecnologie
      - questa tipologia di manutenzione **cambia la natura del prodotto** in modo radicale, mantenendo però tale prodotto

  - <u>La manutenzione è una qualità essenziale del software</u>
    - un prodotto buono è un prodotto aggiornato frequentemente, ovvero in uso manutentivo per un lungo periodo
      - stato manutentivo: tanti utilizzatori del sw, non per obbligo (es. uniweb) ma per scelta (es. Gmail), con l'apporto ad esempio di migliorie al prodotto.

- **Progetti SW fallimentari:**
  - I progetti SW hanno spesso esito insoddisfacente a causa di:
    - difficoltà nelle fasi iniziali
    - ritardi
    - maggior costo
    - cambi di piattaforma e tecnologia necessari ma inattesi
    - difetti residui nel prodotto finale

  - A volte falliscono clamorosamente
    - per obsolescenza prematura
    - per incapacità o impossibilità tecnica di completare
    - per esaurimento dei tempi o dei finanziamenti

  - Per evitare ciò, è necessario:
    - Riuscire a soddisfare obiettivi prefissati entro limiti certi di tempo e di sforzo
    - massimizzare **efficacia** ed **efficienza** (← glossario)

  - Vogliamo applicare principi ingegneristici alla produzione del SW
    - mancano base matematica solida e parametri tecnici certi
    - ciclo virtuoso "esperienza ↔ sistematizzazione"
      - ciò che viene definito "**best practice"** (← glossario)

Fine lezione T1

Lezione T2 – Data: 06/10/2016 – Titolo: **Processi Software**

Definizione di Progetto:

- Consiste nell'**organizzazione di azioni nel tempo, per il perseguimento di uno scopo** predefinito.
- **E' un incarico**, un lavoro che **ha un punto di inizio e fine atteso**, che non è altro che un avanzamento dello stato di vita, all'interno del ciclo di vita del software.
- Il punto chiave è che **esiste un committente:**
  - un progetto **nasce per far avanzare il ciclo di vita** della cosa che consegneremo
  - **assumiamo il ruolo di fornitori**
    - con una responsabilità
    - prendiamo un impegno verso il committente
- Un progetto **evolve quindi nel tempo**, sposta uno stato di vita
- il progetto è sempre **collaborativo**
  - alcune attività possono essere svolte individualmente
- **Tutti i compiti sono pianificati da inizio a fine**
  - **secondo specifici obiettivi e vincoli**
    - I vincoli sono dati:
      - dal tempo disponibile
      - dalle risorse utilizzabili
      - dai risultati attesi
  - la pianificazione è l'identificazione di cosa c'è da fare, e sotto quali vincoli
- **se esiste un progetto, occorre farne gestione**
  - **Vedi lezione T5**
  - **viene istanziato il processo di gestione del progetto**
- Deve essere **quantificabile:**
  - il progetto è dimensionato secondo il tempo persona (**ore/persona**), che **rappresenta il costo del progetto.**
  - le ore/persona vanno spalmati sul calendario; **i punti di inizio e fine sono stabiliti dal contratto con il proponente**.

- Cosa c’è dentro ad un progetto?
  - Secondo **SEMAT**, gli **elementi essenziali di un progetto** (=i suoi “ingredienti”) sono presentati qui di seguito:

    - **Customer:**

      - **Stakeholder** (← glossario)
        - non rappresenta solamente il **cliente**, ma anche l’**utilizzatore**
          - il cliente può ancora non esistere, ad esempio nella Apple, con il telefono senza tastiera, avevamo dei clienti “surrogati”, cioè che non esistevano ancora

      - **Opportunity**:
        - Devo avere l’opportunità, cioè deve valerne la pena, economicamente o intellettualmente
          - quello che mi pagano, mi basta a fare quello che devo fare?
          - quello che imparo, mi servirà? È richiesto nel mercato?

    - **Solution:**

      - **Requirements:**
        - = le richieste
        - devo sapere e capire qual è il problema
          - devo sapere ciò che devo fare
        - sono **originate da:**
          - **opportunità**
            - faccio una cosa perché capisco che ce n’è bisogno
          - **stakeholders**

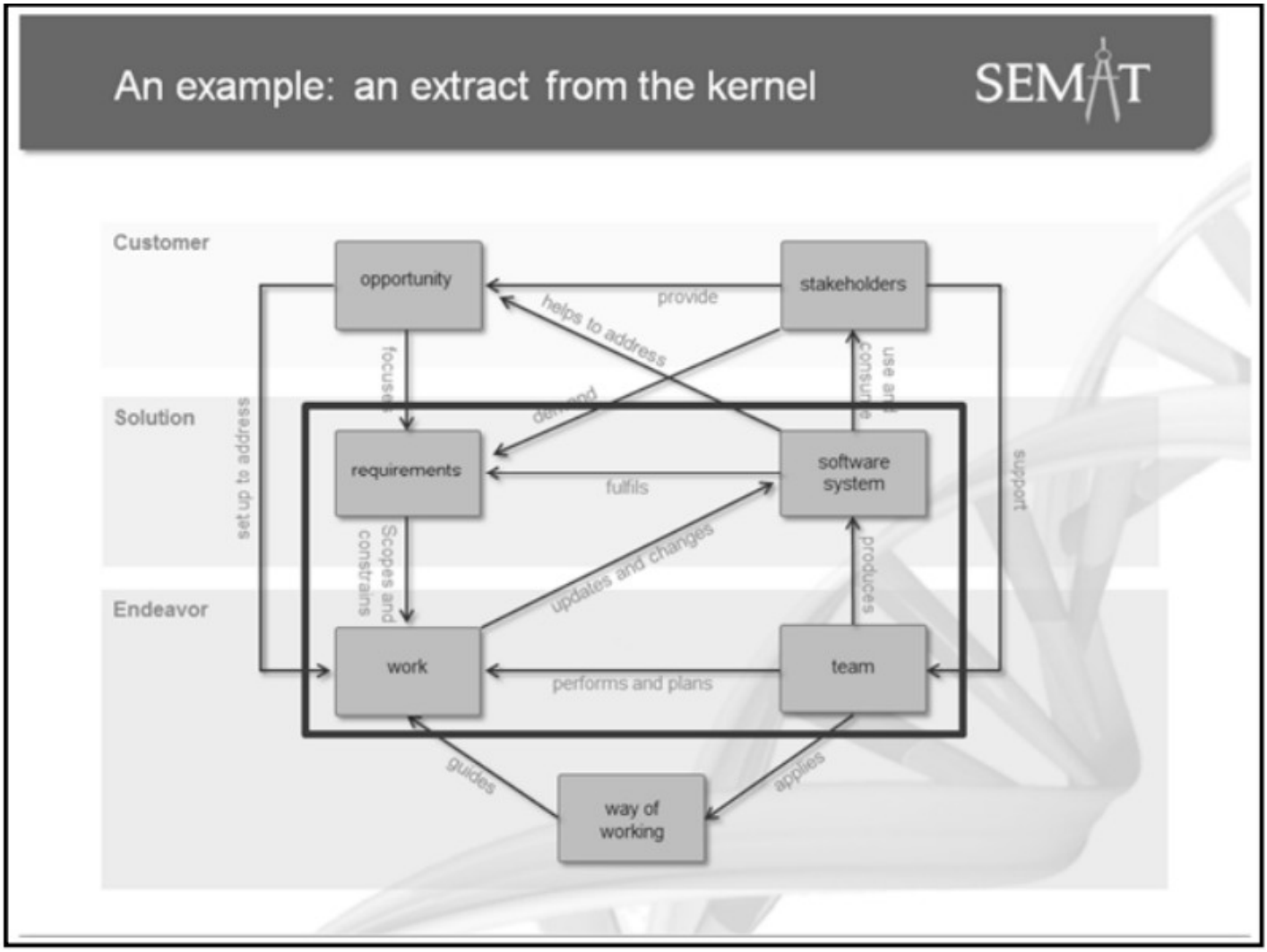
      - **Software System:**
        - devo avere una soluzione in grado di risolvere il problema per poter procedere alla creazione del sistema software

    - **Endeavor:**
      - = impegno volitivo

      - **Team:**
        - il **team** produce **work** (lavoro) <u>utile</u>
        - il work non può essere fatto singolarmente, ha sempre bisogno di un team

      - **Work:**
        - è una produzione utile, non è usa e getta

      - **Way of working:**
        - i progetti per natura, se voglio che abbiano successo, devono avere un modo di lavorare programmato e collaborativo
        - deve essere dunque organizzato
        - è fondamentale, dev’essere alla base di tutto.
        - È essenziale per essere **sistematici**, **disciplinati** e **quantificabili**



fulfils = soddisfa

Le 4 P de SWE

- **P = People**
  - gli stakeholder
- **P = Project**
  - L'insieme delle attività di produzione
- **P = Product**
  - Software e documentazione
- **P = Process**
  - L'organizzazione delle attività

vediamole più da vicino…

- **P = People**
  - **Business Manager**
    - chi fissa gli obiettivi in termini di costi, profitto, priorità strategiche
      - competizione sul mercato, soddisfazione del cliente, efficienza, efficacia
  - **Project Manager**
    - chi gestisce le risorse di progetto e riferisce all'organizzazione e al cliente
  - **Development team**
    - chi realizza il prodotto: il luogo di appartenenza dei **software engineer**
  - **Customers**
    - chi compra il prodotto SW
  - **End users**
    - chi usa il prodotto SW
  - Figure professionali:
    - **Software engineer != Programmatore**
      - **Programmatore:**
        - figura professionale dominante negli anni pionieristici dell'informatica ('50-'70)
        - scrive programmi per se stesso, da solo, sotto la propria responsabilità tecnica
        - svolge un'attività creativa fortemente personalizzata
      - **Software engineer:**
        - realizza parte di un sistema complesso con la consapevolezza che potrà essere usato, completato e modificato da altri
        - deve guardare e comprendere il quadro generale nel quale il sistema cui contribuisce si colloca
          - la dimensione "sistema" include, ma non si limita al SW
        - deve operare compromessi intelligenti e lungimiranti tra visioni e spinte contrapposte
          - costi – qualità
          - risorse (tempo, mezzi, competenze) – disponibilità
          - ecc…

        - Principi etici di SWE
            - considerare la qualità come il primo obiettivo
            - impegnarsi a produrre sw di alta qualità
            - aiutare il cliente a comprendere i suoi veri bisogni
            - adottare i processi più adatti al progetto
            - ridurre la distanza intellettuale tra il sw e il problema da risolvere
            - **essere proattivi** nel cercare e rimuovere gli errori
            - motivare, formare, far crescere le persone
            - un principio chiave: distinguere tra problematiche essenziali… e accidentali
                - **problematiche essenziali:**
                    - riguardano: specifica, realizzazione, verifica, manutenzione di prodotti sw
                    - nessuna soluzione tecnica o tecnologica può esonerarci dall'impegno concettuale, di astrazione, di analisi, di rigore ecc…, necessario per affrontare le problematiche essenziali
                - **problematiche accidentali:**
                    - gli strumenti e le tecniche per la rappresentazione e la verifica di accuratezza di rappresentazione delle problematiche essenziali
                    - l'evoluzione tecnica e tecnologica può rendere più agevole affrontare le problematiche accidentali

- **P = Project**
    - **Pianificazione:**
        - organizzare e controllare tempo, risorse e risultati
    - **Analisi dei requisiti:**
        - definire cosa bisogna fare
    - **Progettazione:**
        - definire come bisogna farlo
    - **Realizzazione:**
        - farlo con la massima efficienza e la massima efficacia
    - **Verifica e Validazione:**
        - assicurare che quanto fatto soddisfi i requisiti e non contenga errori
    - **Manutenzione:**
        - assicurare piena utilizzabilità del software fino al momento del suo ritiro

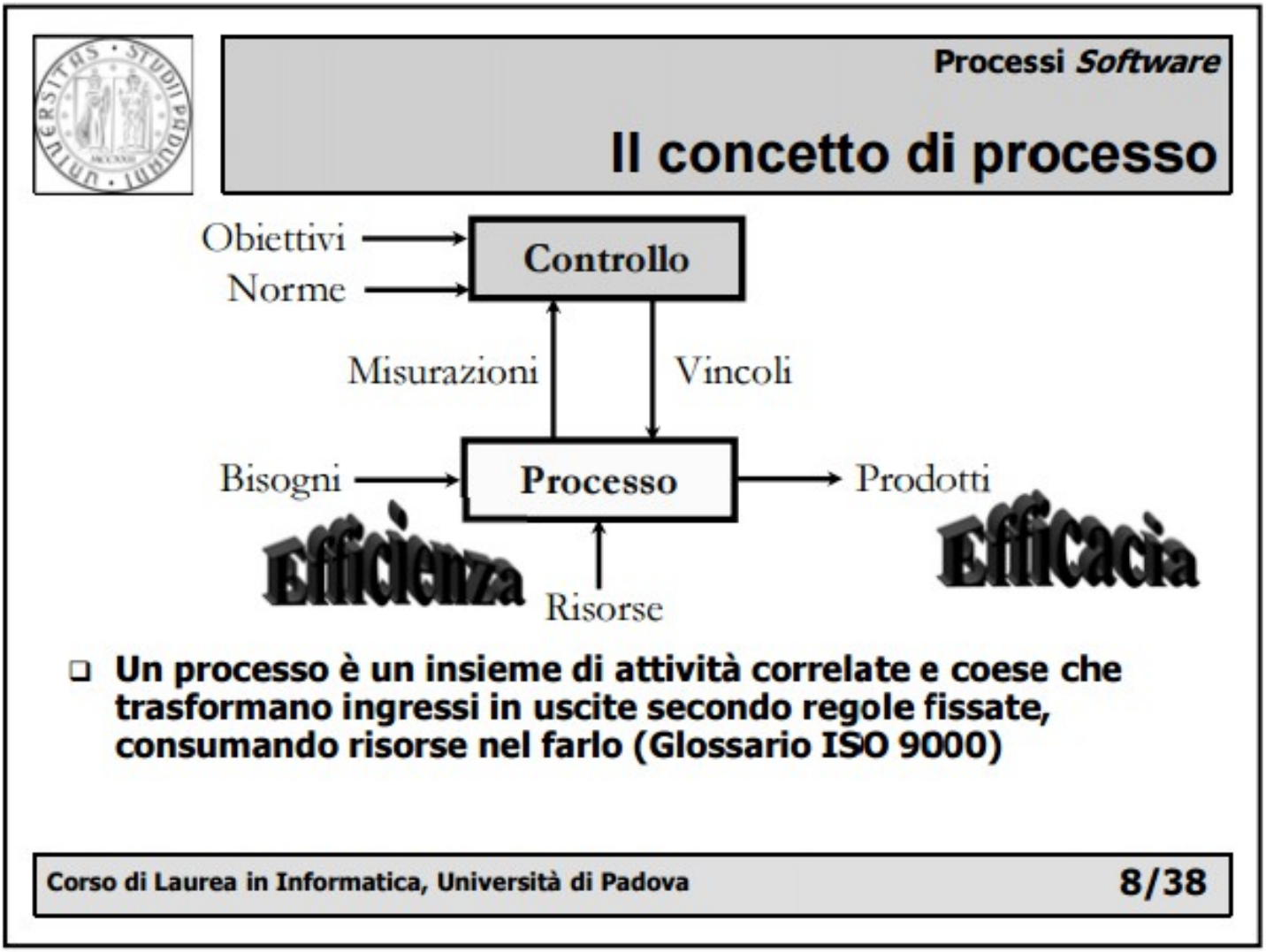
- **P = Process**
    - Sono degli aggregati, ordinati, di attività, il cui unico fine è quello di far transire lo stato del ciclo di vita
        - definisce come tali attività sono correlate le une alle altre
        - come attuarle e in quale ordine
        - i loro gradi di libertà
    - Alimenta svariati modelli di ciclo di vita del sw
        - per organizzare al meglio le attività necessarie all'interno di vincoli dati di tempo, di risorse e di obiettivi

Fine set slide #1 - "INTRODUZIONE"

“PROCESSI SOFTWARE”

Cos'è un processo:

- **Processo di ciclo di vita:**
  - sono degli aggregati, ordinati, di attività, il cui unico fine è quello di far transire lo stato del ciclo di vita di un prodotto software
    - “aggregati” perché un processo coinvolge più persone e quindi dev’essere reso sistematico e disciplinato



Il concetto di processo: (Definizione secondo ISO)

- **Processo:**
  - insieme di attività correlate e coese (=tutte servono) che trasformano ingressi in uscite secondo regole fissate, consumando risorse nel farlo.
  - (nella figura sopra)
    - il processo è qualcosa che sta al piano basso
      - dice: “questo è il modo in cui lavoro”
    - un processo che ha **qualità**, ha un piano alto
      - si chiama **controllo di processo**

- Cosa caratterizza un processo?
  - La capacità di misurarsi secondo **efficacia** ed **efficienza** (← glossario)
    - **Tutti i processi vanno misurati**
      - tutte le attività che caratterizzano il processo, devono poter essere misurate
        - **deve avvenire automaticamente**, attraverso l'utilizzo di strumenti
          - esempio: il log di sistema
      - vogliamo fare misurazioni a scopo di raggiungimento degli obiettivi di efficacia ed efficienza, attraverso meccanismi esclusivamente **push**
        - **paradigma "push"**
          - chi genera l'evento, lo spinge verso le persone che possono essere interessate a tale evento
            - Facebook!
            - Analogia della porta
          - serve un supporto informatico che la propaghi
            - chi fa, propaga
        - **paradigma "pull"**
          - quando mi serve una cosa, devo andare a prendermela
            - Analogia della porta
          - non dobbiamo "chiedere" il lavoro che è stato fatto e a che punto siamo
            - **è tempo perso**

  - **Evitare l'analisi retrospettiva:**
    - distruggo l'efficienza
      - ho consumato (troppe) risorse
    - distruggo l'efficacia
      - quello che ho fatto non va bene
    - <u>mi serve un modo che mi aiuti ad avere degli indicatori della distanza tra dove dovrei essere e dove sono attualmente</u>
    - un esempio: la funzione di navigazione di Google Maps
      - prima calcola il percorso (=pianifica il percorso)
      - poi mi accompagna
        - attraverso l'utilizzo di sensori presenti nel dispositivo, <u>misurando costantemente</u>
          - "smartphone" perché ha sensori per misurare
    - **le misurazioni devono dunque essere prese in modo sistematico**
      - come facciamo ad essere sistematici?
        - **seguendo norme di condotta**
          - **aderendo a standard**
      - se non mi sto muovendo nella giusta direzione, sono in grado di correggere quando non è troppo tardi

- **Gli standard di processo:**
  - è un cuneo che non mi fa scendere, non peggioro
    - mi garantisce un comportamento non meno peggio di… quello che è dato dallo standard
  - si dividono in:
    - **Generali:**
      - **ISO/IEC 12207**
    - **Settoriali:**
      - concepiti per uno specifico dominio applicativo
        - IEC 880: settore nucleare
        - RTCA DO-178B: settore aeronautico
        - ECSS E40: settore spaziale
        - …

  - **ISO/IEC 12207**
    - **è il modello più noto e riferito per lo standard (generale) di processo**
    - modello ad alto livello
      - **identifica:**
        - **i processi dello sviluppo software**
          - descrive i processi in termini di attività e compiti elementari
        - **i prodotti dei processi**
      - specifica le responsabilità sui processi
      - "esplode" i processi in attività

    - I processi vengono raggruppati in **3 categorie:**
      - **processi primari** [→ main]
        - ciascun processo può essere visto come un main
        - governano le cose da fare
      - **processi di supporto** [→ procedure]
        - procedure che chiamo quando ne ho bisogno
      - **processi organizzativi** [→ linux]
        - l'ambiente con il quale il resto può esistere, in modo ragionevole, con un way of working

ISO/IEC 12207 – 2

- **Processi primari:**
  - **esiste un progetto se e solo se è attivo almeno uno dei processi primari**
  - si dividono a loro volta in **5 sottogruppi:**
    - **acquisizione:**
      - gestione dei propri sotto-fornitori
        - cioè di chi ci fornisce una componente del nostro prodotto

    - **fornitura:**
      - dicono quali sono le attività da svolgere se voglio essere un fornitore
      - gestione dei rapporti con il cliente

    - **sviluppo:**
      - non è solo “scrivere codice”
      - ha un numero molto grande di attività:
        - **requirements elicitation** (=sobbollizione)
          - prendo i **requisiti**, al lordo, e ne estraggo il succo, ciò che mi interessa
          - capisco il problema
        - system requirements analysis
        - system architectural **design**
          - to design = progettare = dare forma ad un’idea
        - software requirements analysis
        - software architectural design
        - software detailed **design**
        - **coding and testing**
          - scrittura del codice e testing
        - software integration
        - software **qualification** testing
          - qualification = dimostro che vado bene
            - la risposta alla qualification è una risposta <u>quantificata</u>
            - dimostro di essere conforme
            - dimostro di aver soddisfatto i **requisiti**
        - system integration
        - system **qualification** testing
        - software installation

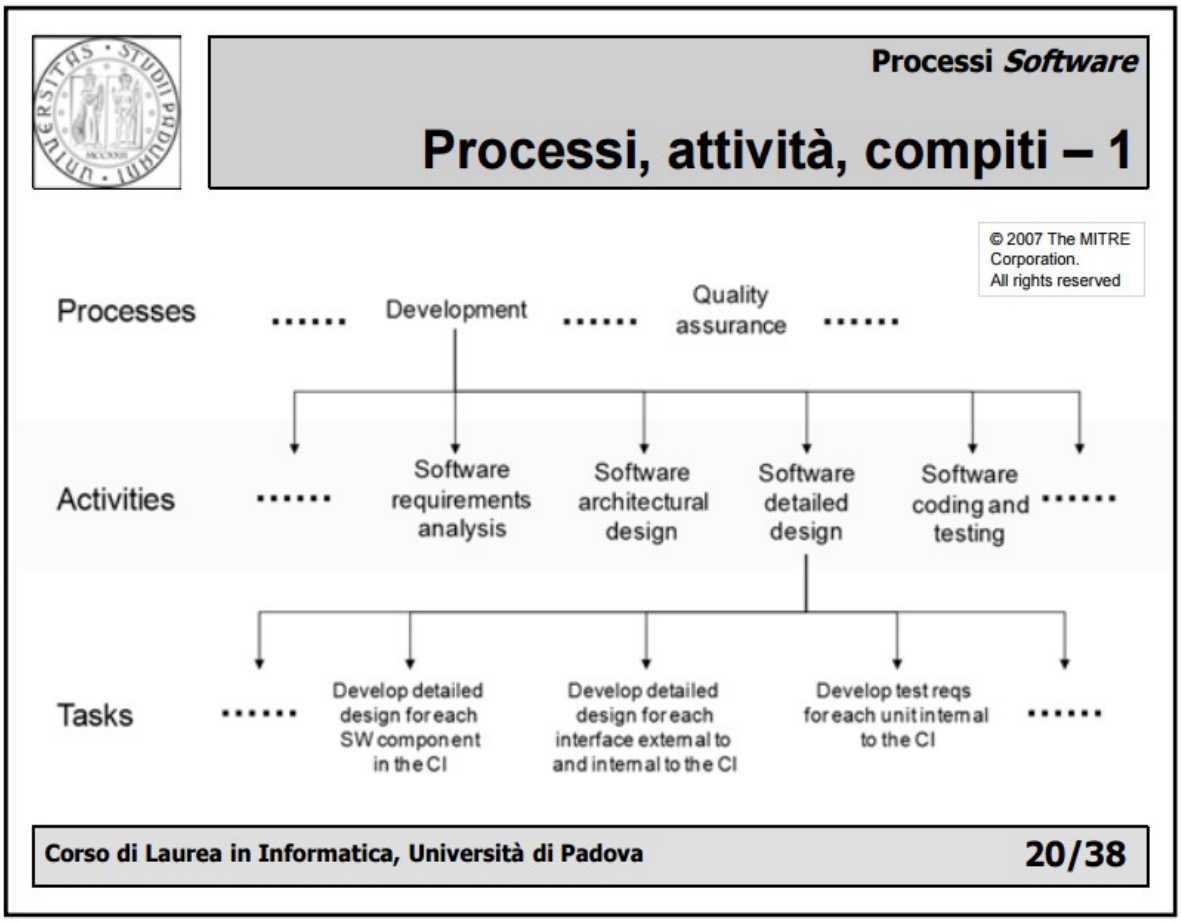
      - il software engineer riesce a fare qualsiasi delle azioni “development” [→ any of this]
      - il software developer svolge una sola singola azione

    - **gestione operativa:**
      - installazione ed erogazione dei prodotti e/o servizi

    - **manutenzione:**
      - correttiva
      - adattiva
      - evolutiva

- **Processi di supporto:**
  - ne fanno parte, i processi di:
    - **documentazione:**
      - è un processo di supporto importantissimo
      - necessario per fare manutenzione
    - **gestione delle versioni e delle configurazioni** (← glossario)
    - **Verifica** (← glossario)
    - **Validazione** (← glossario)
    - accertamento della qualità
    - **revisioni congiunte con il cliente:**
      - **audit (RR, RA)**
    - **verifiche ispettive interne:**
      - **joint review (RP, RQ)**
    - **risoluzione dei problemi**

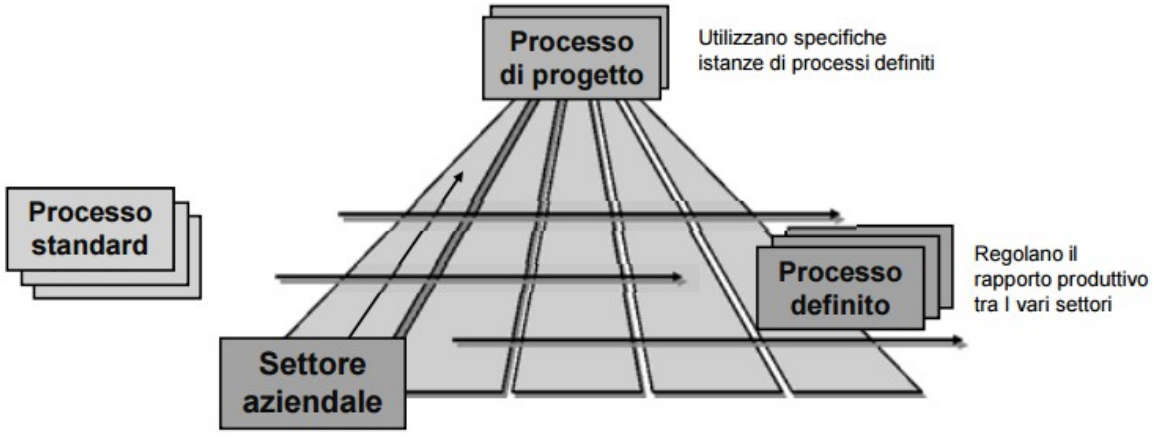
- **Processi organizzativi:**
  - **servono a consentire il lavoro collaborativo**
    - che sia **sistematico, disciplinato e quantificabile**
  - abbiamo processi di:
    - **gestione di progetto:**
      - project management
      - @LezioneT5
    - **gestione delle infrastrutture:**
      - infrastructure
    - **miglioramento del processo:**
      - improvement
      - ci interessa migliorare le cose che farò nuovamente nel futuro
        - cercando di essere più efficiente ed efficace
          - misurandomi
        - voglio, devo, posso migliorare strategia
          - come posso quindi migliorarmi?
            - **PDCA** (← vedi **ciclo di Deming**)
        - nel libero mercato, è meglio avere un <u>miglioramento continuo</u> piuttosto di un miglioramento occasionale
    - **formazione del personale:**
      - training
      - qualunque organizzazione che svolge progetti deve lanciare attività di formazione per poter svolgere progetti
        - la formazione non dev'essere retrospettiva!
          - "ah, si faceva così..."



- Processi, attività, compiti
  - i **processi** sono tra loro relazionati in modo chiaro e distinto
    - **modularità**
  - le **attività** di processo sono ben definite e correlate tra di loro
    - **coesione**

- Relazione tra attività e tecniche:
  - le tecniche sono “ricette” per svolgere determinati compiti
  - vincoli o strategie di sviluppo restringono i gradi di libertà disponibili nello svolgimento

- **Processi software, aziende e progetti:**
  - la buona organizzazione di un'azienda si basa sul riconoscimento dei propri processi, la loro adozione consapevole ed efficace e il loro supporto efficiente
  - **processo standard:**
    - riferimento di base generico
    - condiviso tra aziende diverse nello stesso dominio applicativo
    - è una sorta di template
  - **processo definito:**
    - specializzazione di processo standard
    - per adattarlo alle specifiche esigenze e caratteristiche aziendali
    - chiari, stabili, documentati
    - indipendenti da:
      - modello di **ciclo di vita** adottato
      - tecnologie
      - dominio applicativo
      - documentazione richiesta
  - **processo di progetto:**
    - istanze di processi definiti
    - utilizzano risorse aziendali per raggiungere obiettivi prefissati e limitati nel tempo (progetti)
    - ben pianificati
    - chiare scelte di specializzazione
      - definire lo scenario di applicazione
      - definire attività e compiti aggiuntivi o specifici
      - organizzare le relazioni tra i processi specializzati
      - fattori di specializzazione:
        - dimensione del progetto
        - complessità del progetto
        - rischi identificati
          - dominio applicativo
          - tecnologie in uso
        - competenza ed esperienza delle risorse umane
        - fattori dipendenti dal contratto in essere
    - massima attenzione nel condurre il progetto
      - la prima volta il progetto è "pilota"
    - valutazione critica dell'esito
      - formalizzare e istituzionalizzare le parti che hanno ben operato



- **Organizzazione di processo:**
  - Per essere disciplinati si ha bisogno di una forma di standardizzazione, per "tenere alta" la qualità di un lavoro ripetitivo che rischia continuamente di degradare.
  - L' organizzazione interna dei processi dev'essere incentrata sul principio del miglioramento continuo (W.A. Shewhard, W.E. **Deming**)
    - se voglio rendere un processo migliorativo, non è sufficiente effettuare misurazioni. Servono anche PDCA.

  - **Ciclo di Deming:**
    - Il PDCA (conosciuto anche come "Ciclo di Deming" o "Ciclo di Shewhart") è un metodo per la gestione delle attività di processo ripetibili e misurabili e per la manutenibilità dei processi stessi.
      È un metodo iterativo suddiviso in quattro fasi (Plan-Do-Check-Act, da cui l'acronimo) e assicura un non decremento della qualità ad ogni ciclo.
      Fissati degli obiettivi di miglioramento desiderati si iterano le attività previste dal PDCA, fino al raggiungimento degli stessi.
      I miglioramenti ai quali si fa riferimento sono legati all'efficienza e all'efficacia.
      Migliorare l'efficienza significa usare meno risorse per fare lo stesso lavoro.
      Migliorare l'efficacia significa divenire più conformi alle aspettative.

    - **P = PLAN**
      - Fase di pianificazione del lavoro e degli obiettivi di miglioramento;
    - **D = DO**
      - Viene dunque attuato quanto pianificato nella fase precedente e vengono inoltre raccolti dati;
    - **C = CHECK**
      - Questi dati vengono analizzati e rapportati agli obiettivi pianificati;
    - **A = ACT**
      - Se sono emersi miglioramenti, questi vengono regolamentati e integrati nello standard dell'organizzazione.
        In caso contrario è necessario analizzare gli stati del ciclo per individuare le cause del mancano raggiungimento degli obiettivi stabiliti.
        Si procederà quindi ad una nuova iterazione.

Fine lezione T2

Lezione T3 – Data: 07/10/2016 – Titolo: **Processi Software_parte2**

Ciclo di vita del software:

- **Ciclo di vita del Software:**
  - esiste un ciclo di vita perché esiste un progetto che realizza un prodotto
  - è l'insieme degli stati che il prodotto assume dal concepimento al ritiro
    - gli stati rappresentano il grado di maturazione del prodotto SW
      - **concezione → sviluppo → utilizzo → ritiro**
        - **Concezione:**
          - nasce l'idea e/o il bisogno
            - il cliente che attua il processo di acquisizione ha bisogno di un certo software
            - un produttore di software ha bisogno di un certo software
        - **Sviluppo:**
          - se c'è qualche interessato, si procede allo sviluppo
        - **Utilizzo:**
          - vogliamo degli utilizzatori del software
          - il software è buono se l'utilizzo è duraturo nel tempo
            - soprattutto se il software è libero
            - es. Uniweb non è libero
        - **Ritiro:**
          - per ritiro si intende il momento in cui il prodotto cessa di essere seguito dai creatori

    - gli archi (=transizioni di stato) sono l'insieme di attività (raggruppate in processi) svolte sul prodotto che servono a farlo avanzare nel grado di maturazione
      - per organizzare le attività di processo implicate
        - identifichiamo le dipendenze tra i loro ingressi (input) e le loro uscite (output)
        - fissiamo il loro ordinamento nel tempo e i criteri di completamento e avanzamento

    - La durata temporale entro uno stato di ciclo di vita o in una transizione tra essi viene detta "**FASE**" (← glossario)

    - Specifici obblighi, regole e strategie determinano:
      - la natura degli stati di inizio e fine (cosa contengono)
      - Le pre- e post- condizioni poste sulle transizioni tra gli stati

  - **Fasi** tipiche includono lo studio o analisi, la progettazione (architetturale e di dettaglio), la realizzazione, il collaudo (o testing), la messa a punto, la manutenzione e l'estensione.

  - conoscere il ciclo di vita serve per valutare costi, tempi, obblighi e benefici associati allo svolgimento di un progetto software
    - di questa conoscenza c'è bisogno prima di intraprendere il progetto (analisi preventiva)
      - un'analisi consuntiva non è utile, in quanto ormai è troppo tardi per agire!

- **Modelli di ciclo di vita:**
  - Un **modello è una rappresentazione astratta di come si fanno le cose, associate ad un perché** si fanno le cose
    - l'adozione di un modello **richiede un sistema di qualità per garantire e misurare conformità e maturità**
  - **I processi software, di per sé, non seguono un ordinamento.** Il modello di ciclo di vita **fornisce le relazioni temporali e logiche tra i processi, rispetto agli stati di ciclo di vita.**
    - attorno al modello pianifico, organizzo, eseguo e controllo lo svolgimento delle attività necessarie
  - Il modello di ciclo di vita adottato **pone vincoli su pianificazione e gestione del progetto**
    - precede e non segue la selezione di metodi e strumenti di sviluppo
    - è indipendente da metodi e strumenti di sviluppo
  - Tra le qualità che contraddistinguono l'IS — sistematicità, disciplina, quantificabilità — i modelli di ciclo di vita nascono con l'**obiettivo di perseguire la quantificabilità**, che è la più difficile da soddisfare.

  - esistono **diversi possibili modelli di cicli di vita:**
    - **non diversi per numero e significato di stati**
    - **ma diversi per le transizioni tra essi e le loro regole di attivazione**
    - **si possono dividere in due grandi famiglie:**
      - **sequenziali:**
        - **scompone un progetto per attività**
      - **iterativi:**
        - **scompone un progetto per funzionalità**
        - consentono **maggior capacità di adattamento**
          - evoluzione di problemi, requisiti utente, soluzioni e tecnologie
        - comportano il **rischio di non convergenza**
          - è necessario limitare superiormente il numero di iterazioni
        - incrementale, evolutivo, a spirale, agile, …
    - variano per:
      - tipo di prodotto
      - esigenze
      - tipo di contratto
      - ecc…

- Processi e modelli di ciclo di vita:
  - La specifica dei processi **non** determina la scelta di un modello di ciclo di vita
  - **Il livello di coinvolgimento del cliente determina natura, funzione e sequenza dei processi di revisione necessari**

- **Fattori che influenzano la scelta del modello di ciclo di vita:**
  - **politica di acquisizione e di sviluppo a livello di sistema:**
    - versione unica/multipla
    - dipendenze da/verso altre componenti (parti del sistema)

  - **natura, funzione e sequenza dei processi di revisione richiesti per verificare lo stato di avanzamento**
    - revisioni interne/esterne | bloccanti/non bloccanti

  - **necessità di fornire evidenza preliminare di fattibilità:**
    - Sviluppi prototipali
      - usa e getta / da mantenere / da evolvere
    - Studi e analisi preliminari
      - precedenti l'autorizzazione allo sviluppo

    - **evoluzione del sistema e dei suoi requisiti**
      - possibile necessità di iterazioni
      - particolari esigenze di configurazione di sistema (build, deployment)

Fine set slide #2 - "PROCESSI SOFTWARE"

Fine lezione T3

Lezione T4 – Data: 10/10/2016 – Titolo: **Il ciclo di vita del software**

Evoluzione dei modelli di cicli di vita:

- **“Code-’n-Fix”**
  - Un non-modello.
  - Attività eseguite senza organizzazione preordinata
  - Risulta in progetti caotici non gestiti né gestibili
  - Non ho nessuna sensazione di avanzamento né di convergenza
    - mi sembra di iterare

- Modelli organizzativi (alcuni…)

  - **Modello (di sviluppo) SEQUENZALE:** la catena di montaggio:
    - detto anche:
      - a cascata
      - waterfall
    - definito nel 1970 da Winston W. Royce
    - **concentrato sull’idea di processi ripetibili**
    - l’ordine delle cose è dato da una catena sequenziale, ordinata, nel quale **ho il prodotto finale solo alla fine della catena.**
    - **Successione di fasi** (← glossario) **rigidamente sequenziali**
      - **non ammette ritorno a fasi precedenti**
        - infatti stiamo parlando di “Fase”, e non di “Periodo”
      - **non siamo mai in 2 stati diversi**
      - **fasi distinte e non sovrapposte nel tempo**
      - **il prossimo stato sarà più avanzato**
      - **approccio pesantemente top-down**
      - **ogni fase viene definita in termini di:**
        - **attività previste**
        - **prodotti attesi in ingresso**
        - **prodotti attesi in uscita**
        - **contenuti e struttura dei documenti**
        - **ruoli coinvolti**
        - **scadenze di consegna**

    - **Prodotti:**
      - **principalmente “documenti”**, fino a includere il SW
        - i documenti prodotti da una fase devono essere approvati per il passaggio alla fase successiva (**modello <<*document driven*>>**)

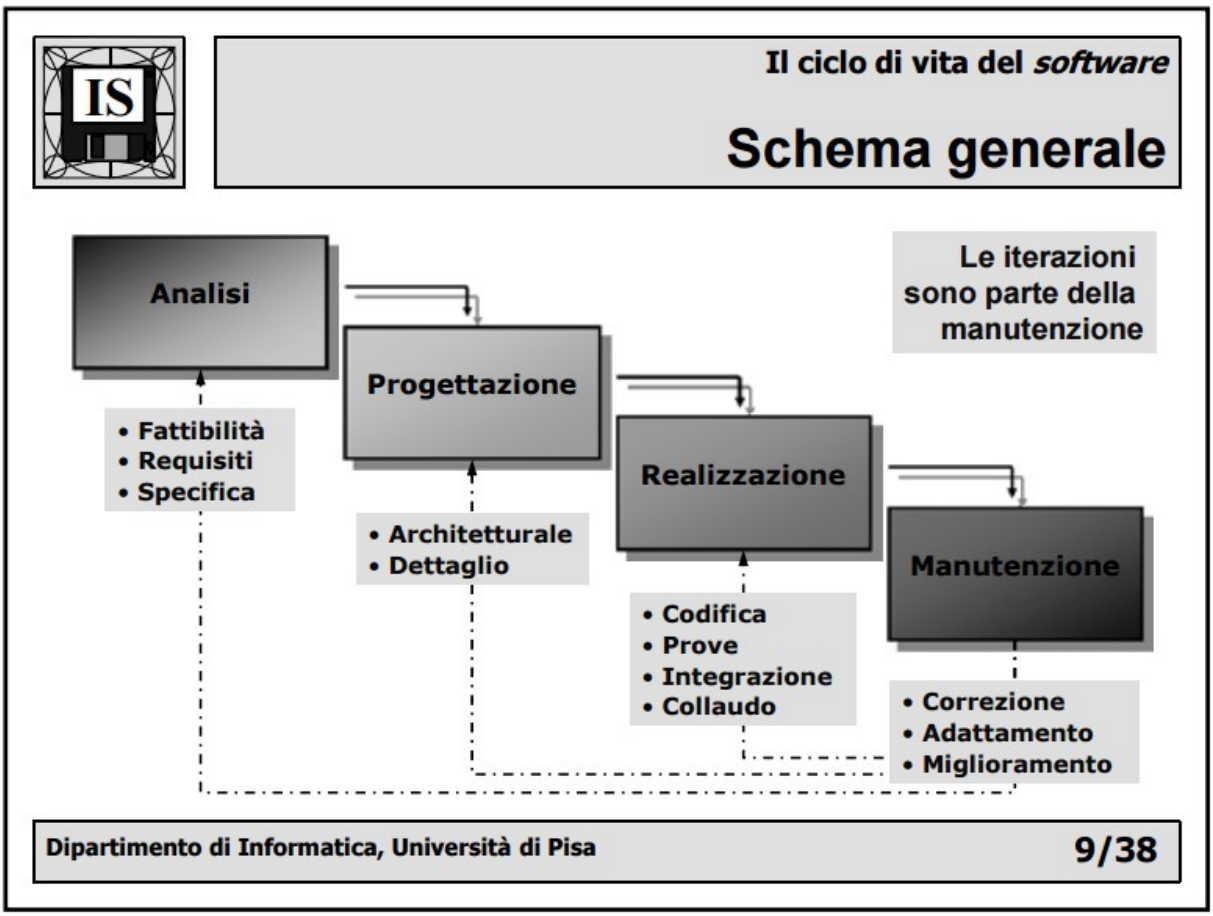
      - Ogni stato è caratterizzato da **pre-condizioni** di ingresso e **post-condizioni** di uscita:
        - si entra in uno stato se e solo se vengono soddisfate le pre-condizioni
        - si esce da uno stato se e solo se vengono soddisfatte le post-condizioni
        - il loro soddisfacimento è dimostrato da prodotti costituiti prima da documentazione e poi da SW

- Questo modello, **esplode lo stato "sviluppo" del ciclo di vita in 4 stati**:
  - **Analisi:**
    - è la prima cosa da fare
    - **capire il problema**
    - **ho una possibile soluzione del problema→ transizione**

  - **Progettazione:**
    - insieme di attività che rendono concreto il concetto precedentemente fissato
    - **penso a come le parti del mio sistema possano essere realizzate, come le varie parti interagiscono tra loro.**
      - non sto ancora scrivendo codice
    - **ho identificato (progettato) tali parti → transizione**

  - **Realizzazione:**
    - **programmo ciò che ho progettato**
    - il programmatore non è un creativo
    - **il software è stato programmato e collaudato → transizione**

  - **Manutenzione**



- **Adatto allo sviluppo di sistemi complessi** sul piano organizzativo
  - le iterazioni costano troppo per essere un buon mezzo di mitigazione dei rischi tramite approssimazioni successive

- Correttivi del modello sequenziale:
  - **Difetto principale: eccessiva rigidità**
    - stretta sequenzialità tra fasi
    - richiede molta manutenzione
    - esprime una visione burocratica e poco realistica
    - è un modo di procedere molto lento
    - funziona solo se c'è un buon raccordo tra chi spiega come fare e chi farà
    - **l'acquirente del software dovrà sapere dall'inizio ciò che vuole**
      - non ammette modifiche nei requisiti in corso d'opera
  - **Correttivo 1: prototipazione:**
    - vengono introdotti prototipi di tipo "usa e getta"
      - solo per capire meglio i requisiti
  - **Correttivo 2: cascata con ritorni:**
    - ammette il ritorno ad uno stato precedente
    - **Ritorni: iterazione o incremento?**
      - non sempre gli stakeholders comprendono dall'inizio ogni aspetto del sistema richiesto
        - se il problema è molto complesso conviene prevedere iterazioni
          - le iterazioni possono essere distruttive e eliminare e rimpiazzare lavoro precedente
      - non sempre è desiderabile rimandare alle fasi finali l'integrazione di tutte le parti del sistema (big bang)
        - in tal caso è meglio l'integrazione successiva di piccole parti
          - questo è un procedimento incrementale
      - iterazione e incremento coincidono quando la sostituzione raffina ma non ha impatto sul resto

- **Il modello sequenziale segue un approccio predittivo**
  - **basato cioè su piani che devono essere rispettati**
    - **permette di stimare con precisione una data di consegna e un preventivo**

- **Modello (di sviluppo) INCREMENTALE:**
  - costrutti per iterare:
    - ciclo for (**Incremento** (← glossario))
      - quando sò il numero di iterazioni necessarie
        - è quantificabile
      - ad ogni passo so che sono avanzato
    - ciclo while (**Iterazione** (← glossario))
      - quando non riesco a dire prima quando finirò
      - procedere per iterazioni significa operare raffinamenti o rivisitazioni
      - non è quantificabile
      - non so se convergo

  - **Per superare le difficoltà del modello sequenziale ibrido, nacque il modello incrementale: qui i cicli non sono più iterazioni ma incrementi;**
    - **procedere per incremento significa aggiungere ad un impianto base**
      - **sviluppo un pezzo, lo metto assieme agli altri, guardo se va bene con gli altri e nel caso proseguo**
        - development system increment
        - validate increment
        - integrate increment
        - validate system

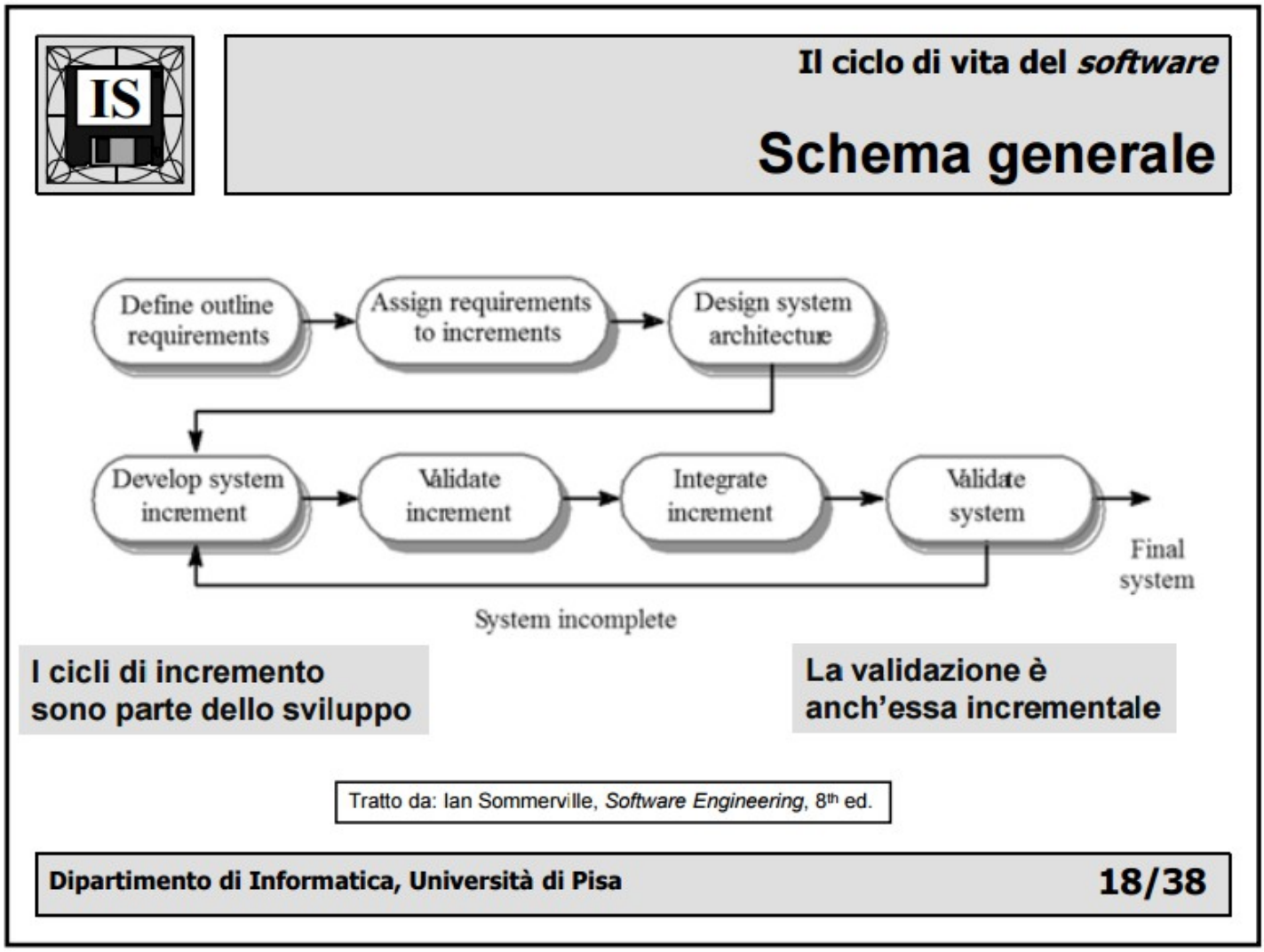
  - **E' un modello funzionante solo quanto il cliente è molto consapevole, sa quello che vuole, essendo comunque più "gentile" del modello sequenziale**

  - **Vantaggi dei modelli incrementali:**
    - **possono produrre "valore" ad ogni incremento**
      - un insieme non vuoto di funzionalità diventa presto disponibile
    - **ogni incremento riduce il rischio di fallimento:**
      - senza però azzerarlo a causa dei costi aggiuntivi derivanti dalle eventuali iterazioni
      - **le funzionalità essenziali sono sviluppate nei primi incrementi**
        - i requisiti utente sono quindi classificati e trattati in base alla loro importanza strategica
        - attraversano quindi più fasi di verifica
          - e quindi diventano più stabili con ciascuna iterazione

  - **Analisi e progettazione architetturale <u>non</u> ripetute**
    - nella variante Staged Delivery

  - **Il modello incrementale segue un approccio adattativo:**
    - la realtà è considerata inerentemente imprevedibile.
    - sarebbe preferibile un approccio predittivo, che permette di stimare con precisione una data di consegna e un preventivo; tuttavia **l'approccio adattativo può essere conveniente in uno scenario in cui i requisiti cambiano in corso d'opera.**

- **Schema generale del modello incrementale:**
  - **Definisci i contorni dei requisiti**
    - capisci il problema
  - **Assegna i requisiti agli incrementi**
    - pensa ad una sequenza, in modo tale da avere l'impressione che avanzando mi renda conto sempre più di avere una buona soluzione, di convergere
  - **Progetta l'architettura del sistema**
  - **Sviluppa il primo incremento**
    - inizio a codificare
  - **Valida l'incremento**
    - controlla che soddisfi i requisiti identificati
  - **Integra l'incremento prodotto**
    - se c'era qualcosa, aggiungi quanto prodotto a quello che c'era prima
  - **Valida il sistema**
    - controlla che quello che hai messo insieme funzioni
  - **Ritorna indietro e sviluppa il prossimo incremento**
    - fino a quando il sistema non diventerà finale



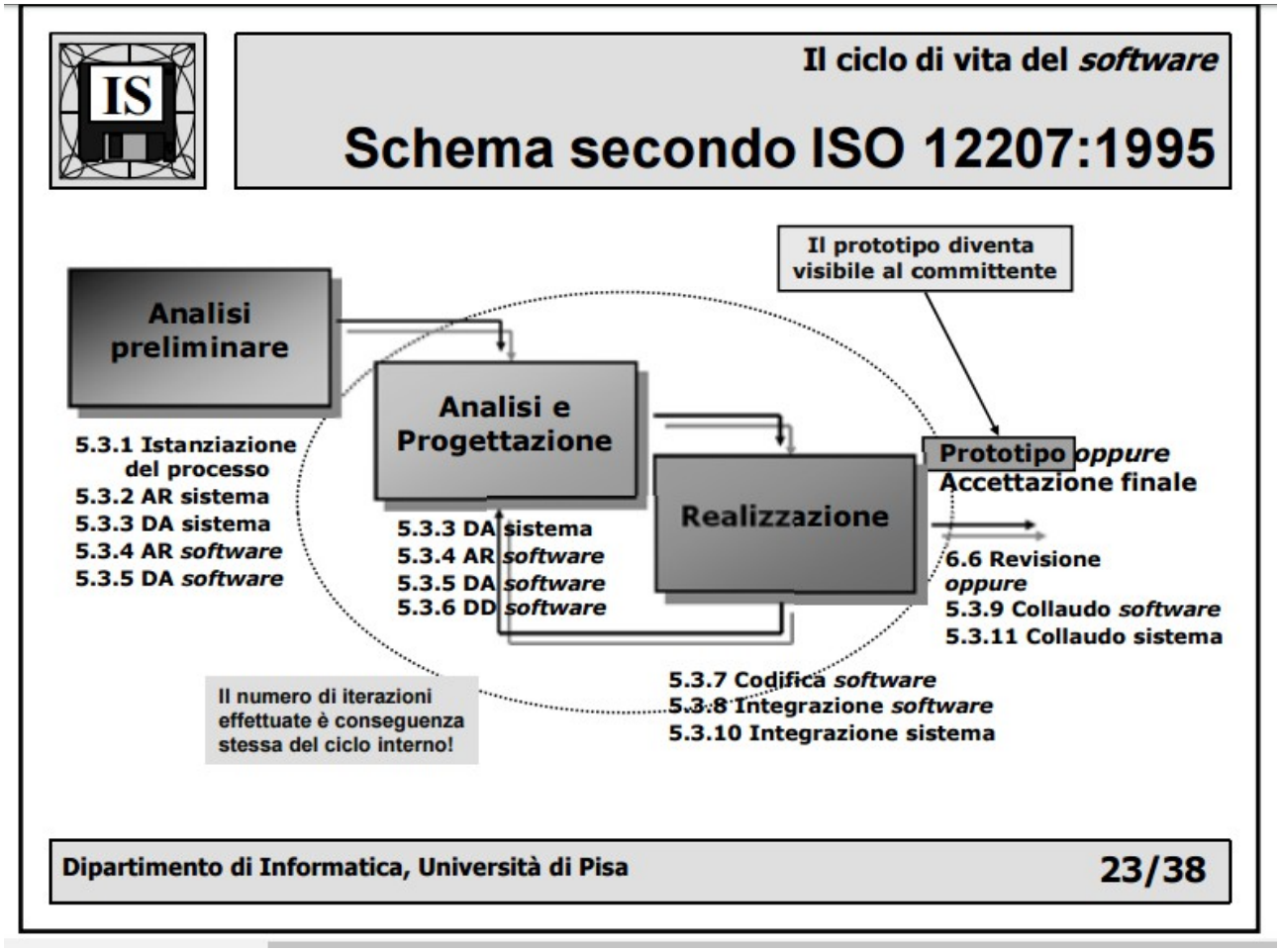
- **Modello (di sviluppo) EVOLUTIVO:**
  - **è un modello incrementale in cui gli incrementi successivi sono versioni (prototipi) usabili dal cliente**

  - può richiedere **il rilascio e il mantenimento di più versioni del prodotto simultaneamente disponibili**
    - utenti possono voler tenere versioni precedenti di un certo software
      - es: microsoft

  - **aiuta a rispondere a bisogni non inizialmente preventivabili**

  - **ogni fase ammette iterazioni multiple**

  - **Schema generale:**
    - **Analisi preliminare:**
      - per identificare i requisiti di massima
      - per definire l'architettura di massima
      - per pianificare i passi di analisi e realizzazione evolutiva
    - **Analisi e realizzazione di una evoluzione:**
      - per raffinamento ed estensione dell'analisi
      - per progettazione, codifica, prove e integrazione
    - **Rilascio di "prototipi", poi accettazione finale**



- **Modello (di sviluppo) A SPIRALE:**
  - proposto da Barry W Boehm
    - IEEE Computer, maggio 1988

  - **per miglior controllo del rischio di progetto**
    - è il rischio maggiore
    - pone grande attenzione sugli aspetti gestionali
      - pianificazione delle fasi
      - analisi dei rischi (**modello <<risk driven>>**)

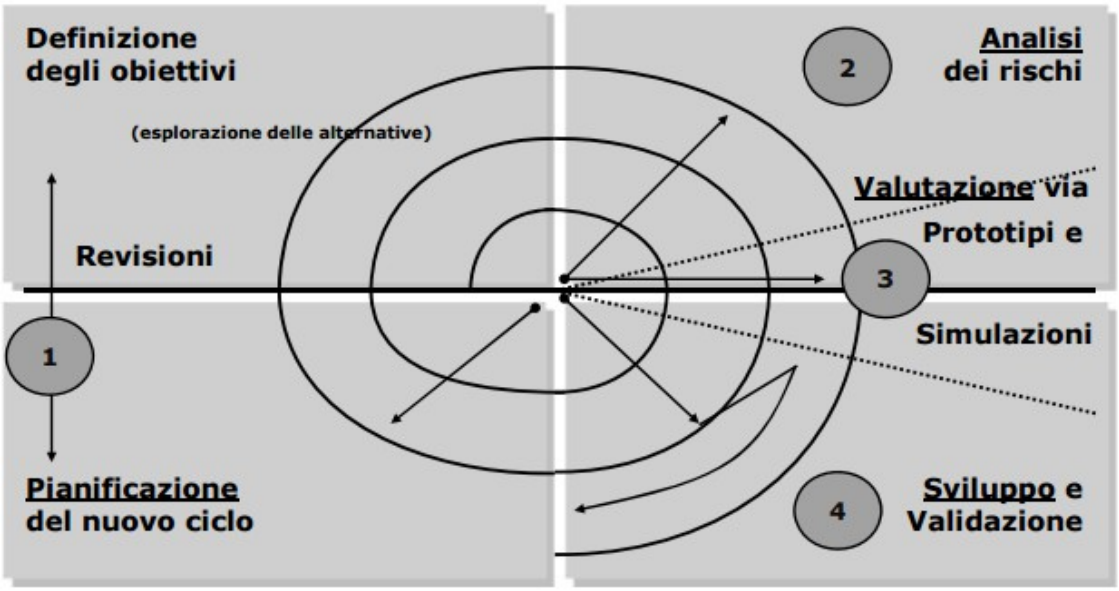
  - **questo modello è utilizzato in ambito di ricerca**, ma non in ambito dello sviluppo software.

  - **Lo sviluppo procede a cicli inizialmente rapidi ma via via sempre più lenti:**
    - nella spirale:
      - parto dal centro
        - cicli interni rapidi e ripetuti
          - dedicati ad analisi e sviluppi prototipali
      - i cicli esterni sono così lenti che possono aderire ciascuno ad un diverso modello di ciclo di vita
    - **ad ogni ciclo si analizzano i rischi e si compiono simulazioni.**
    - misura del successo di un progetto è il diametro della spirale.

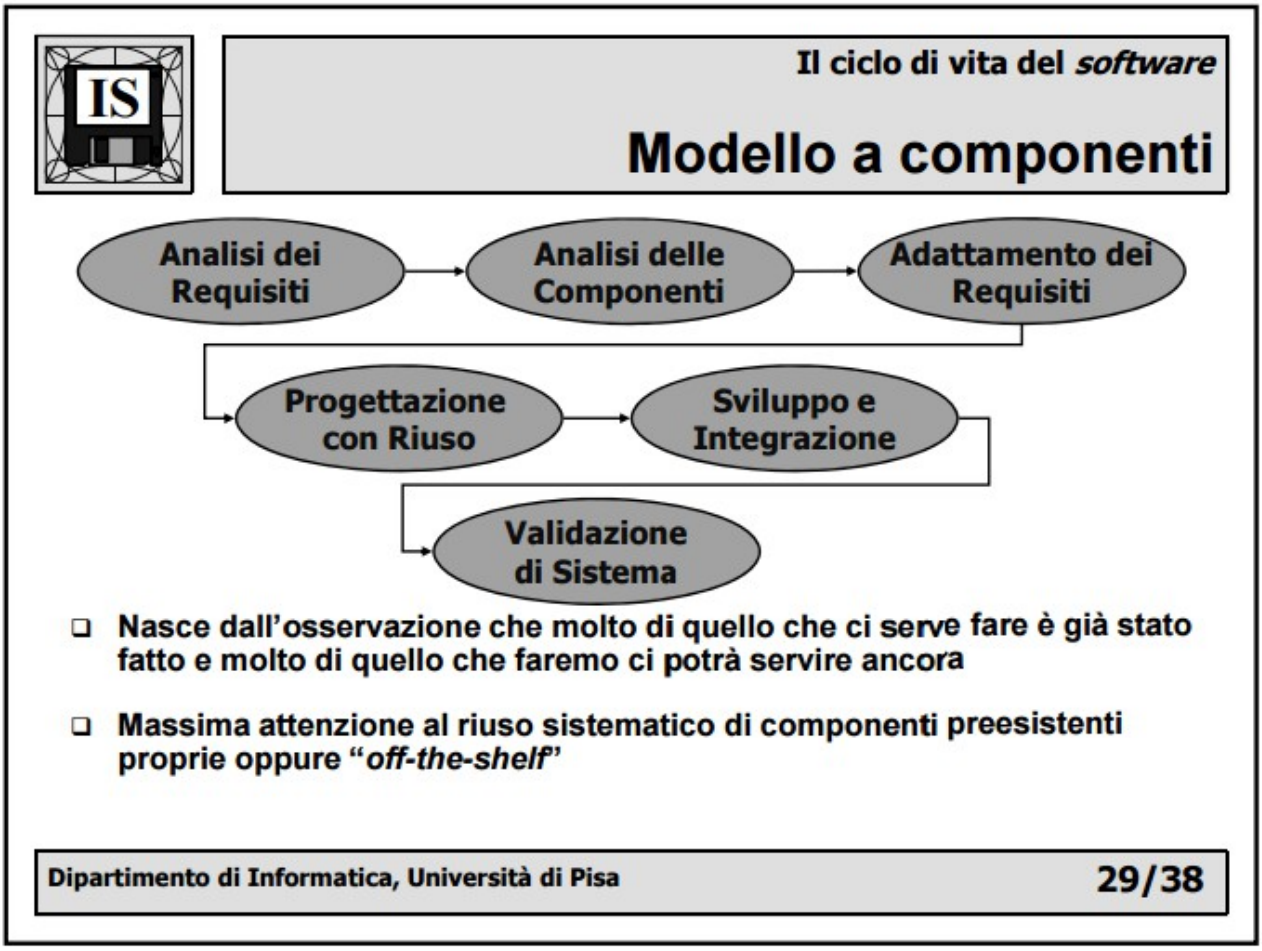
  - **richiede forte interazione tra committente e fornitore:**
    - committente:
      - definisce gli obiettivi
      - definisce i vincoli sulla pianificazione
    - fornitore:
      - sviluppo e validazione
    - entrambi:
      - svolgono l'analisi dei rischi

  - prevede **quattro macro-fasi**:
    - **definizione degli obiettivi**
    - **analisi dei rischi:**
      - studio delle conseguenze
      - valutazione delle alternative con l’ausilio di prototipi e simulazioni
    - **sviluppo e validazione:**
      - realizzazione del prodotto
    - **pianificazione della successiva iterazione**

Schema generale



- **Modello (di sviluppo) A COMPONENTI:**
  - **nasce dall'osservazione che molto di quello che ci serve fare è già stato fatto e molto di quello che faremo ci potrà servire ancora**
  - **parto dai bisogni**
  - **faccio un inventario di ciò che già ho**, che potrebbe essere utile a soddisfare il bisogno
    - **eventualmente negoziando sui requisiti**, in modo da poter riutilizzare il software
  - **massima attenzione al riuso sistematico di componenti preesistenti** proprie oppure "off-the-shelf" (=pronto all'uso)
  - **progettazione con riuso**, e tale riuso mi costringe a soddisfare determinati vincoli



- **Modello (di sviluppo) AGILE:**
  - **dinamico, a cicli iterativi e incrementali**

  - **CAOS**

  - nascono alla fine del ‘90 come **reazione alla eccessiva rigidità** dei modelli allora in vigore

  - si basano su **quattro principi fondanti:**
    - 1) le persone e le interazioni sono più importanti rispetto ai processi e agli strumenti
      - INUTILE
    - 2) molto più importante il software rispetto alla documentazione
      - INUTILE, per la manutenzione mi serve la documentazione!
    - 3) meglio ingaggiare una collaborazione con il cliente rispetto ad un contratto
      - cioè, l’interazione con gli stakeholder va incentivata e non ingessata
      - SI e NO
    - 4) rispondere al cambiamento anziché agire secondo un piano
      - INUTILE (e dannoso), meglio seguire un piano

  - l**’idea base è il concetto di “user story”:**
    - **un compito significativo che l’utente vuole svolgere con il software richiesto**
    - **ogni “user story” è definita da:**
      - **un documento di descrizione**
      - **la minuta di conversazioni con il cliente** (gli stakeholder in generale) **per fissare la comprensione comune**
      - **la strategia da usare per confermare che il software realizzato soddisfi gli obiettivi**

  - **assunti base:**
    - è possibile **suddividere il lavoro da fare in piccoli incrementi a valore aggiunto che possono essere sviluppati indipendentemente**

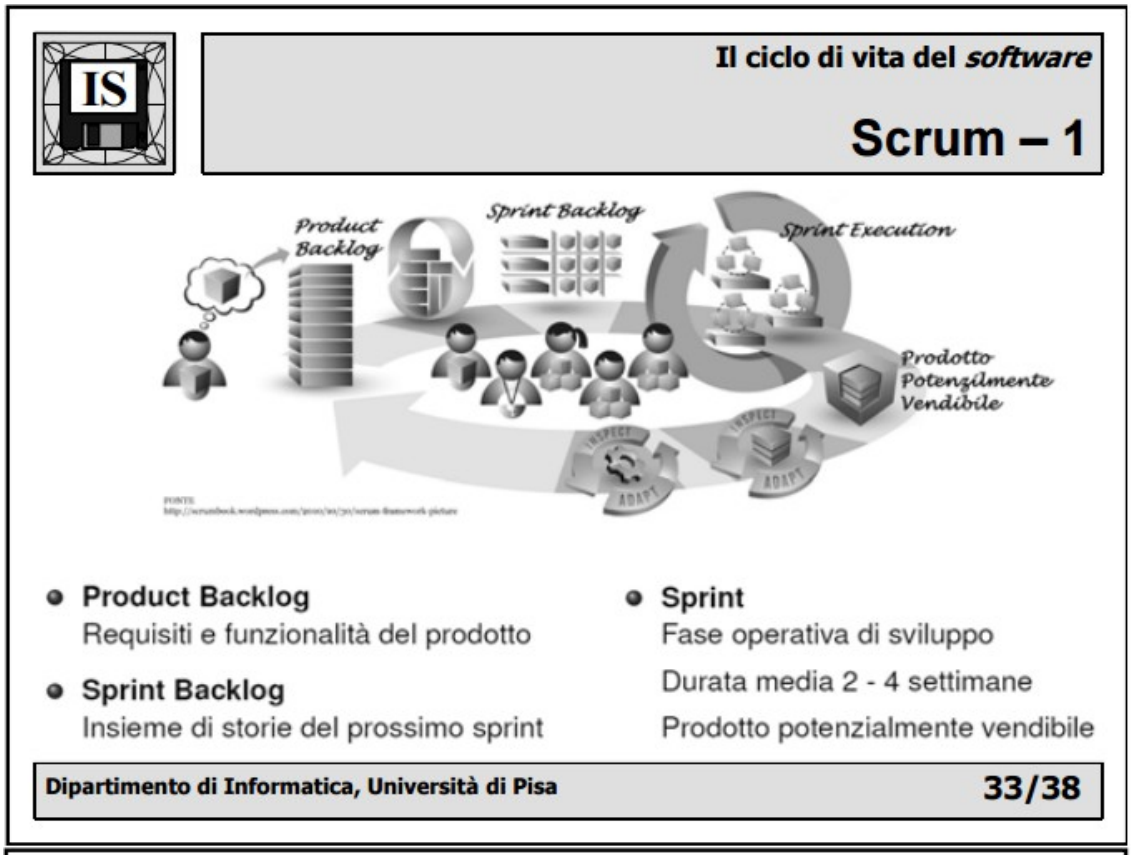
  - **obiettivi strategici:**
    - **poter costantemente dimostrare al cliente quanto è stato fatto**
    - verificare l’avanzamento tramite progresso reale
    - **dare agli sviluppatori la soddisfazione del risultato**
    - assicurare che l’intero prodotto software è ben integrato e verificato

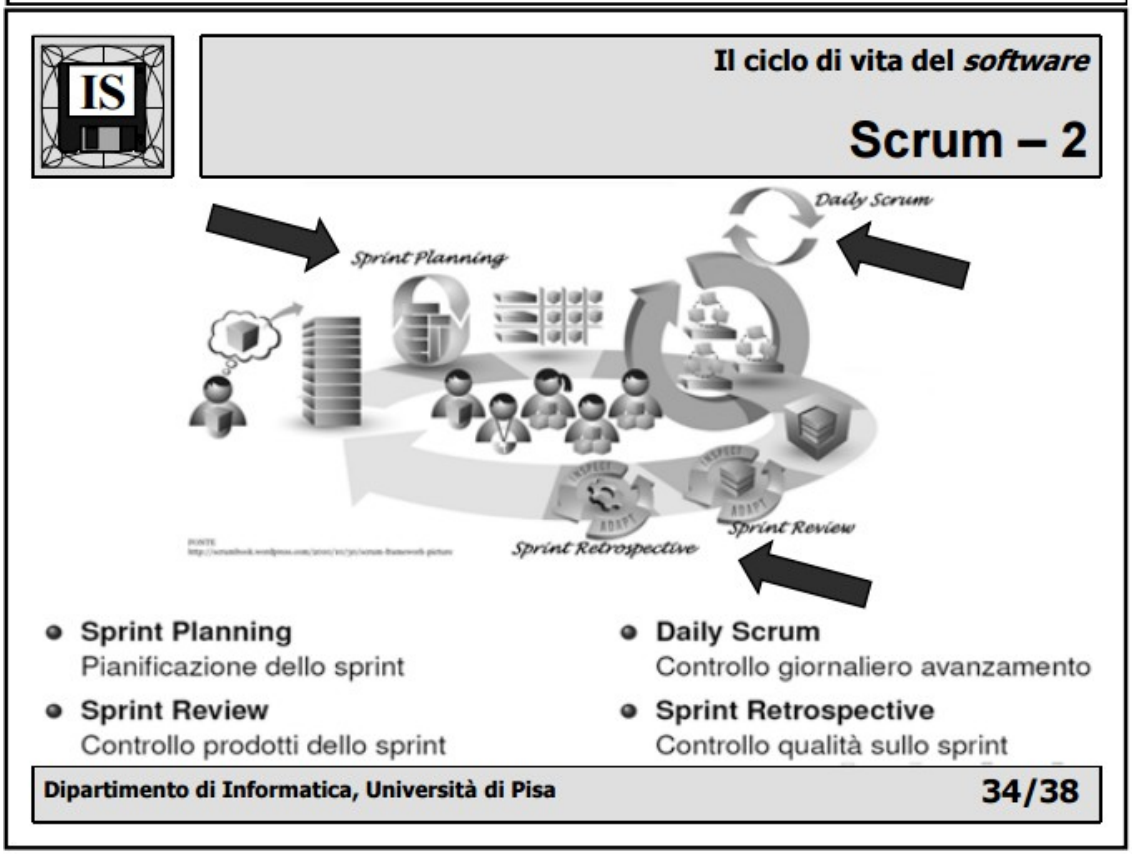
- esempi:
  - Kanban (just-in-time)
  - Scrumban
  - **SCRUM (caos organizzato):**
    - approccio apparentemente confuso, ma dietro ad una logica forte e governata
    - **richiede <u>molta</u> esperienza**
    - immagine mentale: **lavagna con post-it** di cose da fare, piccoli segmenti del progetto
      - si chiama **backlog:**
        - "un mucchio di cose che farò"
        - **taglio un albero (=progetto) ed ogni ceppo (log) lo butto dietro e lo userò**
      - serve una persona (**product owner) molto preparata e con esperienza** (owner) **per creare e mantenere la backlog**
        - deve essere dunque **in grado di individuare tutte le azioni piccole e individualmente assegnabili** che serviranno tutte per sviluppare il prodotto
      - la pila di cose da fare potrebbe potenzialmente crescere
      - **non prendiamo "post-it" a caso, ma le cose da fare hanno una priorità e chiamiamo ciò "sprint" (= fase operativa di sviluppo)**
        - il **product owner** individua gli **sprint**
        - i**n un periodo di tempo ristretto (2-4 settimane), le persone incaricate dovranno prendere questi sprint, svilupparli e integrarli al prodotto**
          - **ottenendo degli incrementi**

    - **assomiglia ad un modello incrementale, ma con 2 differenze:**
      - **1) il product backlog è fatto di pezzi di programmazione**
        - sono così piccole **da poter essere portate a termine da una singola persona**
      - **2) gli incrementi che produco sono tutti funzionanti** (compilati ed eseguiti) **anche se manca il resto**

    - **controllo quotidiano dell'avanzamento dei lavori**
      - **stand up meeting**

    - **finito lo sprint** (= fatto l'incremento), **se ne fa una revisione critica**
      - **se è andato male, potrebbe mettere in discussione il backlog iniziale**
        - **sprint retrospective**
      - più lungo è lo sprint, più incerto è l'esito
        - quindi **meglio avere sprint piccoli**





Fine set slide #3 - “IL CICLO DI VITA DEL SOFTWARE”

Fine lezione T4

Lezione T5 – Data: 13/10/2016 – Titolo: **Gestione di progetto**

Gestione di progetto:

- **E’ un processo organizzativo**

- Come tutte le attività che vogliamo legare all'IS, avrà le seguenti caratteristiche:
  - **sistematico**
  - **disciplinato**
  - **quantificabile**

- **<u>La gestione di progetto</u> è compito del Responsabile di Progetto, e consiste di:**
  - **istanziare i processi nel progetto**
  - **stimare costi e risorse necessarie**
  - **pianificare le attività e assegnarle alle persone**
  - **controllare le attività e verificarne i risultati**

- **Ruoli:**
  - allocare risorse per un progetto significa assegnare attività a ruoli, e ruoli alle persone
  - ogni persona in azienda ha un **ruolo, che può essere di 4 tipi:**
    - **Sviluppo:**
      - responsabilità tecnica e realizzativa
    - **Direzione:**
      - responsabilità decisionale
    - **Amministrazione:**
      - gestione dei processi
    - **Qualità:**
      - gestione della qualità

- **Profili professionali:**
  - Ogni persona ha un profilo professionale, cioè un insieme di competenze (tecnologiche e metodologiche) e un'esperienza (espressa in anni e partecipazione a progetti) che fanno da requisiti per l'assunzione di un ruolo in un progetto.

  - **Responsabile:**
    - **è il Project Manager**
    - **approva l'offerta** e i relativi allegati
    - **partecipa al progetto per tutta la sua durata**
    - **rappresenta il progetto presso il fornitore e presso il committente**
    - **approva l'emissione di documenti**
    - **redige Organigramma e Piano di Progetto**
      - elabora ed emana piani e scadenze
      - coordina le attività del gruppo
    - fa fondamentalmente **3 cose:**
      - **pianifica**
      - **gestisce le risorse umane:**
        - gestisce quindi i ruoli degli altri
          - mi servono X analisti, dal al…
      - **tiene il corso della situazione, del progresso**

  - **Amministratore:**
    - **E' responsabile dell'efficienza e dell'operatività dell'ambiente di sviluppo**
    - **E' responsabile della <u>redazione e attuazione di piani e procedure di Gestione per la Qualità</u>**
    - **Collabora alla redazione del Piano di Progetto**
    - **Redige le Norme di Progetto** per conto del Responsabile
    - **Controlla l'ambiente di lavoro:**
      - **predispone gli strumenti**
      - **gestisce la repository** (← glossario)**:**
        - **Controlla versioni e configurazioni del prodotto**
        - **Gestisce l'archivio della documentazione di progetto**
      - amministra le risorse e le infrastrutture
      - **risolve problemi legati alla gestione dei processi**
      - **non compie scelte gestionali**
        - attua le scelte tecnologiche concordate con i responsabili aziendali e di progetto

- **Analista:**
  - è responsabile delle **attività di analisi:**
    - **devono capire bene il problema**
      - devono raccontarlo in termini utili e capibili da chi farà poi lo sviluppo
      - hanno grande impatto sul successo del progetto
  - **redige lo Studio di Fattibilità e l'Analisi dei Requisiti**
  - **sono pochi e raramente seguono il progetto fino alla conclusione**
  - **in un modello di ciclo di vita sequenziale:**
    - sono **attivi dal "tempo 0" al tempo "abbiamo capito il problema"**
    - **analista e progettista non sono attivi simultaneamente**
  - **non sviluppa la soluzione** (→ progettista) ma dice qual è il problema

- **Progettista:**
  - E' responsabile delle **attività di progettazione:**
    - **sviluppano la soluzione**
      - hanno forte influenza sugli aspetti tecnici e tecnologici del progetto
  - **Redige Specifica Tecnica, Definizione di Prodotto e la <u>parte programmatica del Piano di Qualifica</u>**
  - **sono pochi e talvolta seguono il prodotto fino alla manutenzione**

- **Programmatore:**
  - E' responsabile delle **attività di codifica per la realizzazione del prodotto e delle componenti di ausilio necessarie per l'esecuzione delle prove di verifica e validazione**
  - **puri esecutori**
    - partecipano alla realizzazione e manutenzione del prodotto
    - svolgono un compito che qualcun altro ha pensato
  - **hanno competenze tecniche, visione e responsabilità circoscritte**

- **Verificatore:**
  - **E' responsabile delle attività di verifica e validazione**
    - che insieme prendono il nome di qualifica
  - **Redige la parte retrospettiva del Piano di Qualifica**
    - illustra l'esito e la completezza delle verifiche e delle prove effettuate secondo il piano
  - **partecipano all'intero ciclo di vita**
    - si assicurano che quanto fatto soddisfi le attese
  - hanno competenze tecniche, esperienza di progetto, conoscenza delle norme

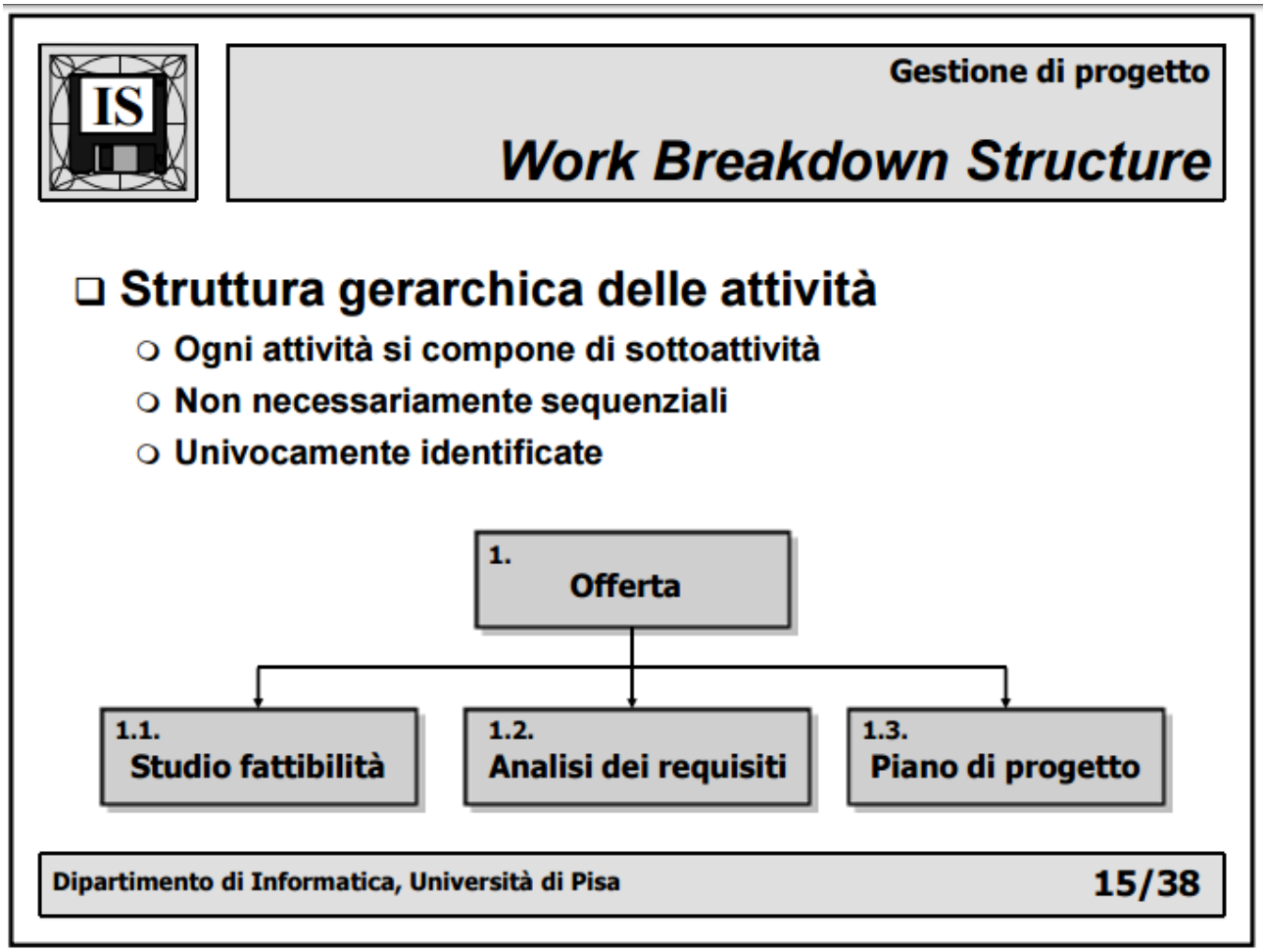
- **Controllore della qualità:**
  - **è una funzione aziendale, e non un ruolo di progetto**
  - serve per dire: stiamo producendo qualità sufficiente o no?
    - prima che sia troppo tardi
    - **dimensioni di qualità:**
      - **dei prodotti e dei processi**
      - sia verso il committente che verso la direzione aziendale

Pianificazione di progetto:

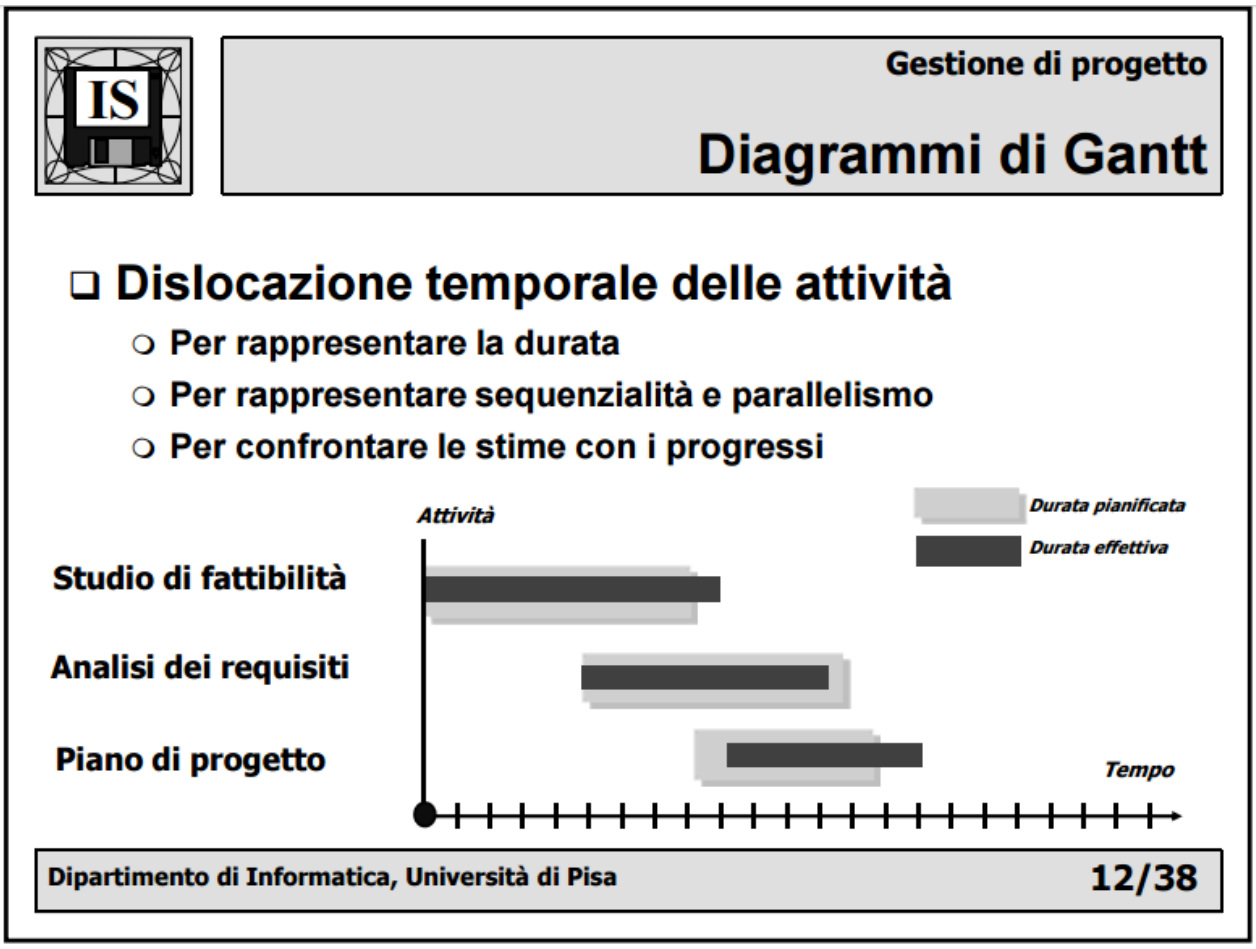
- **Rappresenta il compito più importante del Responsabile di Progetto**
- E' bene notare come lo stato di avanzamento di un prodotto sia rilevante solamente se dà informazioni sulla pianificazione
- **Cosa, quando, chi deve fare le cose:**
  - **il punto invalicabile è la data di fine, ed è da questa data che la nostra pianificazione dovrebbe partire**
    - non dal tempo 0
    - comprimiamo così le cose da fare in modo tale che finiscano non oltre la data di fine

- Strumenti per la pianificazione: **3 strumenti:**

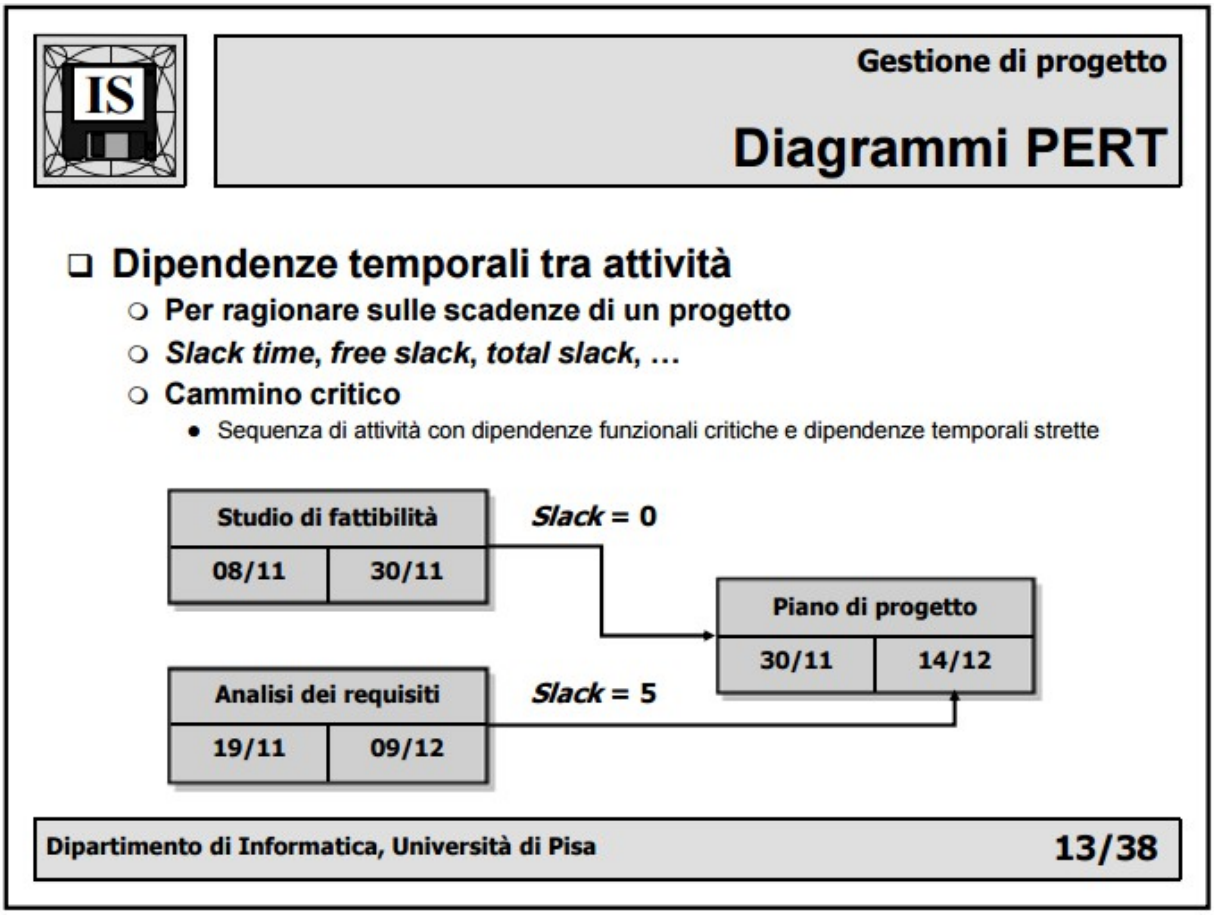
  - **WBS – Work Breakdown Structure:**
    - **con questi diagrammi frantumiamo atomicamente il lavoro:**
      - per fare in modo che i “post-it” siano individualmente realizzabili/attribuibili
    - **il lavoro viene decomposto in modo gerarchico:**
      - **ogni attività si compone di sotto-attività**
        - **fortemente coese**
        - **non necessariamente sequenziali**
        - **univocamente identificate**



- **Diagrammi di Gantt:**
  - **ideali per mostrare la durata, la sequenzialità e il parallelismo delle attività**
  - **permettono di confrontare facilmente le stime con i progressi effettivi**
    - **la durata effettiva di un certo compito è verificabile solo alla fine (consuntivo)**
    - **il diagramma di Gantt è dunque potenzialmente anche uno strumento di consuntivo** (barre scure in figura)
  - **non è buono per gestire le criticità:**
    - **ad esempio per le dipendenze tra le attività**
  - **chi fa cosa e quando:**
    - **asse x = tempo**
      - **in giorni**
    - **asse y = attività**
      - hanno un tempo di inizio
      - hanno un tempo di fine
      - ...e quindi **hanno una durata**
      - **possono essere:**
        - **sovrapposte**
        - **dipendenti**
  - **è uno strumento utile per capire inoltre quante persone serviranno:**
    - le linee orizzontali indicano le persone, anziché le attività
    - **il numero di persone necessario è dato dal momento di massimo parallelismo**



- **PERT – Programme Evaluation and Review Technique:**
  - **Unificano le due tecniche precedenti**
  - **Sono ideali per rappresentare le dipendenze temporali tra attività**
    - **e quindi per ragionare sulle scadenze di progetto**
  - **Il diagramma consiste in un grafo orientato**
    - **gli archi rappresentano le attività**
    - **i nodi sono degli eventi**
      - ogni evento ha una data minima e una data massima
        - la differenza tra questi due tempi è detta slack time
          - rappresenta dunque la quantità di tempo in cui un evento può essere ritardato senza influenzare l'andamento del progetto
  - Nell'immagine: l'arco di connessione serve a dire "l'attività dovrebbe finire prima rispetto a quella da cui dipende
    - lo slack rappresenta una distanza, che è una fonte di problema e anche una fonte di misurazione
    - 1° SLACK = 0
      - dipenza sull'inizio
        - porta dalla fine all'inizio
      - distanza 0 giorni
        - non spreco tempo
        - <u>forte rischio</u>
    - 2° SLACK = 5
      - join sulla fine
        - porta dalla fine alla fine



Stima dei costi di progetto:

- Un altro **compito importante del Responsabile di Progetto** consiste nello stimare i costi
  - in particolare, deve **stimare il tempo/persona:**
    - generalmente settimane/persona o mesi/persona
    - **a questo fine, è utile ricordarsi la legge di Parkinson:**
      - "work expands to fill the time available"
        - il lavoro si espande fino a occupare tutto il tempo disponibile; più è il tempo e più il lavoro sembra importante e impegnativo
        - **più tempo a disposizione si avrà, più se ne sprecherà;** quando il tempo scarseggia, chi lavora lo fa con maggiore efficacia, motivato dal rischio di non riuscire a completare un compito a scadenza ravvicinata, con la prospettiva di possibili conseguenze negative
        - come critica della regolamentazione fine a se stessa

    - **CoCoMo**
      - **Constructive Cost Model**
      - **è uno strumento per la stime del tempo/persona**
        - **basandosi su caratteristiche relative alla quantità di software da produrre**
      - **Assume modello sequenziale e sviluppo da 0**
      - esprime la misura in Mesi/Persona (M/P)
      - **$M/P = C \times D^{S} \times M$**
        - C = fattore di complessità (del progetto)
        - D = misura (in KDSI) della dimensione stimata del prodotto software
          - KDSI = Kilo delivered source instructions, migliaia di righe di codice
        - S = fattore di complessità (del codice)
        - M = moltiplicatori di costo

Rischi di progetto:

- I risultati dei progetti SW, possono portare a:
  - **costi eccessivi**
  - **scadenze non rispettate**
  - **prodotti insoddisfacenti**

- **Un buon modo per gestire i rischi:**
  - nel piano del progetto...
    - **identificazione dei rischi**
    - **analisi dei rischi:**
      - valutazione della probabilità di occorrenza
      - valutazione delle conseguenze, secondo una priorità
    - **pianificazione di come evitare i rischi:**
      - verifica continua del livello di rischio
        - da effettuare su base regolare per determinare il livello corrente di rischio
          - non tutti i rischi sono costanti nel tempo
        - anche per valutare se gli effetti dei rischi possano essere cambiati
          - non tutti gli effetti sono costanti nel tempo
        - riportare periodicamente ciascun rischio serio all'attenzione del management
      - riconoscimento, trattamento e aggiornamento strategie
  - ...durante il progetto
    - **controllo:**
      - attenzione continua tramite rilevazione di indicatori

- **Fonti di rischio:**
  - tecnologie
  - rapporti interpersonali
  - organizzazione del lavoro
  - requisiti e rapporti con gli stakeholder
  - tempi e costi

Piano di Progetto:

- **Il Piano di Progetto fissa:**
  - **le risorse disponibili**
  - **la suddivisione delle attività**
  - **il calendario delle attività**

- **Obiettivi:**
  - **è un documento di strategia**
    - è legato all'**obiettivo di efficacia nel Piano di Progetto**
  - **fissare "milestone"** (←glossario)
  - **il Piano di Progetto:**
    - dice come faccio le cose, con quali tempi, con quali risorse, con quali costi
    - ha come obiettivo il raggiungimento della massima **efficienza ed efficacia** (← glossario)
    - **è un documento che va aggiornato, evolve nel tempo**

- **Struttura del Piano di Progetto nel nostro progetto:**
  - **Introduzione:**
    - scopo e struttura
  - **Analisi dei rischi:**
    - In questa sezione vengono descritti i rischi che potrebbero verificarsi durante lo svolgimento del progetto.
      - I rischi possono essere a livello:
        - tecnologico
        - personale
        - organizzativo
          - errata pianificazione dell'uso delle risorse
        - dei requisiti
          - errata o incompleta analisi dei requisiti
  - **Pianificazione:**
    - Introduzione:
      - Scelta modello di sviluppo
      - Scelta consegna RP (min/max)
      - **Periodi** e **milestones**
        - vengono fissate le milestones
    - Assegnazione delle attività ai periodi:
      - Analisi
      - Progettazione logica
      - Progettazione di dettaglio, codifica e validazione
      - Finalizzazione
  - **Preventivo:**
    - Introduzione
      - In questa sezione viene descritta la suddivisione del lavoro in vari periodi e l'assegnazione ai membri del gruppo in base ai ruoli ricoperti.
    - Dettaglio periodi
    - Totale non rendicontato
    - Totale complessivo
    - Totale rendicontato
  - **Consuntivo:**
    - Introduzione
      - In questa sezione viene presentato il bilancio orario ed economico del progetto a consuntivo. Per ogni periodo viene steso un consuntivo che mostra il quantitativo di ore rendicontate investite durante quel periodo e i totali (rendicontati, non rendicontati e complessivi) delle ore spese fino al termine del periodo preso in esame. Al termine del progetto verrà presentato un consuntivo finale.
    - Consuntivi di periodo
    - Totale non rendicontato
    - Totale rendicontato
    - Considerazioni finali sul consuntivo

- **Preventivo a finire:**
  - Introduzione
    - In questa sezione viene rivisto il preventivo per i rimanenti periodi e il preventivo totale sulla base delle eventuali variazioni esposte nel consuntivo di fine periodo.
  - Dettaglio Periodi
  - Totale non rendicontato
  - Totale complessivo
  - Totale rendicontato

- **Organigramma**

Fine set slide #4 - “GESTIONE DI PROGETTO”

Fine lezione T5

Lezione T6 – Data: 14/10/2016 – Titolo: **Amministrazione di progetto**

Flipped classroom: studio autonomo e presentazione in aula di **strumenti di:**

- **Pianificazione:**
  - presentati in classe:
    - Tom’s planner
  - usati nel nostro progetto:
    - **Teamwork**

- **Coordinamento (collaborazione)**
  - presentati in classe:
    - Wrike
    - Asana
  - usati nel nostro progetto:
    - **Slack**
    - **Teamwork**
    - **Dropbox**
    - **Google Drive**

- **Configurazione** (← vedi parole chiave Lezione T2)
  - presentati in classe:
    - Git
    - GitHub
    - GitLab
  - usati nel nostro progetto:
    - **Git**
    - **GitHub**

- **Versionamento** (← vedi parole chiave)
  - presentati in classe:
    - Git
    - GitHub
    - GitLab
  - usati nel nostro progetto:
    - **Git**
    - **GitHub**

Amministrazione di progetto:

- **Amministratore di progetto:**
  - **E' responsabile dell'efficienza e dell'operatività dell'ambiente di sviluppo**
  - **E' responsabile della redazione e attuazione di piani e procedure di Gestione per la Qualità**
  - **Collabora alla redazione del Piano di Progetto**
  - **Redige le Norme di Progetto** per conto del Responsabile di Progetto
  - **Controlla l'ambiente di lavoro:**
    - **predispone gli strumenti**
    - **gestisce la repository** (← glossario)**:**
      - **Controlla versioni e configurazioni del prodotto** (← glossario)
      - **Gestisce l'archivio della documentazione di progetto**
    - amministra le risorse e le infrastrutture
    - **risolve problemi legati alla gestione dei processi**
    - **non compie scelte gestionali**
      - attua le scelte tecnologiche concordate con i responsabili aziendali e di progetto

- **Documentazione di Progetto:**
  - uno dei compiti dell'amministratore è quello di gestire la documentazione di progetto

    - **disponibilità e diffusione:**
      - I documenti sono utili se e solo se sono sempre disponibili:
        - chiaramente identificati
        - corretti nei contenuti
        - verificati e approvati
        - aggiornati, datati e dotati di versione

      - La loro diffusione deve essere controllata:
        - i destinatari devono essere chiaramente identificati
          - ogni documento ha una sua lista di distribuzione

    - **la documentazione raccoglie tutto ciò che documenta le attività, e si suddivide in due categorie:**
      - **documenti di sviluppo:**
        - documentazione fornita dal cliente
        - diagrammi di progettazione
        - codice
        - piani di qualifica e risultati delle prove
        - documentazione di accompagnamento del prodotto

      - **documenti di gestione del progetto:**
        - documenti contrattuali
        - piani e consuntivi delle attività
        - piani di qualità

    - **ogni documento contiene un diario delle modifiche, in cui vengono riportate le modifiche attuate al documento** rispetto alle versioni precedenti del documento
      - **dev'essere sintetico**, con riferimenti numerici (es: paragrafi modificati)

- **Ambiente di lavoro:**
  - **l'amministratore di progetto gestisce l'ambiente di lavoro:**
    - l’ambiente è **fatto da:**
      - **persone**
      - **ruoli e procedure**
      - **infrastruttura**
        - cioè li risorse hardware e software del progetto

  - **la sua qualità determina la produttività del progetto:**
    - influisce sulla qualità del processo e sulla qualità del prodotto
    - dev’essere:
      - completo:
        - offrire tutto il necessario per svolgere le attività previste
      - ordinato:
        - è facile trovare ciò che vi si cerca
      - aggiornato:
        - il materiale obsoleto non deve causare intralcio

- **Gestione delle modifiche:**
  - un progetto non è esente da richieste di modifiche, che **possono avere origina da:**
    - **utenti**
      - difetti o mancanze
    - **sviluppatori**
      - difetti o mancanze
    - **competizione**
      - valore aggiunto
  - **le richieste di modifiche vanno sottoposte ad un rigoroso processo di analisi, decisione, realizzazione e verifica**
    - change request:
      - autore, motivo, urgenza
      - stima di fattibilità, valutazione di impatto, stima di costo
      - decisione del responsabile
  - **le richieste devono essere tutte tracciabili**
    - **tramite issue tracking** (← glossario) **o ticketing**
    - con uno stato corrente ed eventuale esito chiusura

- **Supporto ai processi:**
  - esempi di processi supportati dall'attività di amministrazione di progetto:

  - **Gestione di progetto:**
    - **pianificazione, stima e controllo dei costi:**

    - **allocazione e gestione delle risorse:**
      - **tramite redazione e consultazione di diagrammi di Gantt e PERT**

    - **strumenti collaborativi di controllo gestionale e di qualità e di coordinamento attività**

  - **Gestione documentale:**
    - Google Docs
    - Versionamento e Configurazione

  - **Analisi e progettazione:**
    - analisi, gestione e tracciamento dei requisiti
    - supporto alle metodologie
      - UML

  - **Codifica e integrazione:**
    - per ambienti integrati di sviluppo
      - ad esempio eclipse
    - strumenti di integrazione continua (continuous integration)
    - misurazione e analisi del codice prima dell’integrazione
    - generazione ed esecuzione automatica dei test prima dell’integrazione

Norme di Progetto:

- **Le Norme di Progetto fissano:**
  - **il way of working** (← glossario)
  - **norme inerenti alle comunicazioni** interne ed esterne al gruppo
  - **norme inerenti alla stesura dei documenti**
  - **norme inerenti alla stesura del codice**
  - **la modalità di lavoro durante il ciclo di vita del progetto**
  - **norme inerenti all'organizzazione dell'ambiente di lavoro e di sviluppo**

- **Obiettivi:**
  - Sono uno strumento operativo di complemento alle procedure
  - **Garantire uniformità nel materiale prodotto**
  - **Favorire la cooperazione tra i membri del gruppo**
  - **Raggiungere il miglior rapporto tra efficacia ed efficienza** (← glossario)**.**
  - **Definisce il way of working:**

- **Chi è il consumatore** delle norme di progetto?
  - **Tutti!**

- **Qual è la sua struttura?**
  - suddivido la struttura **per ruoli?**
    - **No:**
      - **le attività di un ruolo attraversano** più gruppi, ovvero **più processi**
  - **suddivido secondo uno standard:**
    - **secondo processi e attività in esso contenuti**
      - **normiamo le attività:**
        - **attraverso regole il più possibile spiegate in termini di**
          - **motivazione**
            - perché lo facciamo
          - **strumenti che automatizzino il loro svolgimento**

- **Organizzazione di una norma:**
  - **individuiamo due categorie di norme:**
  - **regole:**
    - **convenzioni di cui si riconosce necessità e convenienza**
    - **sottoposte a verifica**
    - **sono utili se sono facilmente verificabili e applicabili, di facile attuazione**
    - **troppe regole sono di difficile attuazione e verifica**
    - **differenza tra regola e procedura:**
      - **regola: comandamento. si fa così.**
        - esempio: si pranza a l'1.
      - **procedura: modo con il quale saremo sicuri, seguendola, che si farà così**
        - esempio: devo fare così per essere sicuro di pranzare a l'1.
  - **raccomandazioni:**
    - **prassi desiderabile, suggerimenti**
    - **senza verifica**

- **Il documento:**
  - **le norme sono utili prima di fare qualcosa**
    - nascono quindi presto
  - sarà un **documento incrementale**
    - l’ingrediente base è il piano di progetto
      - mi dà delle scadenze
  - **viene redatto dall’amministratore**, per conto del responsabile
  - **è un documento interno all’organizzazione**

- **Struttura delle Norme di Progetto nel nostro progetto: (come da standard ISO 12207)**
  - **Introduzione:**
    - scopo e struttura

  - **Processi primari:**
    - **Fornitura:**
      - Scopo:
        - Determinare le procedure e le risorse necessarie allo svolgimento del progetto.
      - Attività:
        - Studio di fattibilità
        - Contrattazione

    - **Sviluppo:**
      - Scopo:
        - Contiene le attività necessarie a produrre il prodotto software richiesto.
      - Attività:
        - Analisi dei requisiti
        - Progettazione
        - Codifica

  - **Processi di supporto:**
    - **Documentazione:**
      - Scopo:
        - redigere e mantenere la documentazione durante l'intero ciclo di vita del software

    - **Gestione della configurazione:**
      - Scopo:
        - (← glossario)

    - **Verifica:**
      - Scopo:
        - garantire che ogni attività dei processi svolti non introduca errori nel prodotto e che soddisfi i requisiti o le condizioni necessarie per essere considerata accettabile.

    - **Validazione:**
      - Scopo:
        - determinare in maniera oggettiva che il prodotto esaminato sia conforme ai requisiti richiesti e che soddisfi il compito per cui è stato creato.

- **Processi organizzativi:**
  - **Gestione dei processi:**
    - Scopo:
      - migliorare l’organizzazione e la cooperazione tra i membri del gruppo

  - **Gestione delle infrastrutture:**
    - Scopo:
      - stabilire e mantenere le infrastrutture e gli strumenti necessari allo svolgimento dei processi durante lo svolgimento del progetto.

  - **Apprendimento:**
    - Scopo:
      - garantire che ogni membro del gruppo abbia conoscenze e capacità sufficienti per svolgere le attività assegnatagli.

Fine set slide #5 - “AMMINISTRAZIONE DI PROGETTO”

Fine lezione T6

Lezione T7 – Data: 20/10/2016 – Titolo: **Analisi dei requisiti**

INGEGNERIA DEI REQUISITI

- **Requisito:**
  - E' una **caratteristica software che dovrà essere messa a contratto con il cliente**
  - **Descrive cosa il sistema deve fare**, cioè i servizi che offre e i vincoli sul suo funzionamento
  - **dev'essere tracciabile all'origine, da una fonte**
    - **l'incontro tra bisogno e soluzione viene chiamato tracciamento** (← glossario)
      - serve al verificatore

  - **Ha due significati:**
    - **significato espresso come bisogno:**
      - è una **condizione necessaria ad un utente per risolvere un problema o raggiungere un obiettivo**
        - **non c'è mai un requisito che non risponde alla domanda perché**

    - **significato espresso come soluzione:**
      - **condizione che deve essere soddisfatta da un sistema per adempiere ad un obbligo**

  - **Classificazioni dei requisiti:**
    - **Requisiti utente e requisiti di sistema:**
      - **requisiti utente: indicano una richiesta generale, ad alto livello**
      - **requisiti di sistema: indicano una definizione formale e dettagliata di una funzione del sistema**

    - **Requisiti di prodotto e di processo:**
      - **requisiti di prodotto: bisogni o vincoli sul prodotto software da sviluppare**
        - risponde alla domanda: cosa devo fare?
      - **requisiti di processo: vincoli sullo sviluppo del software**
        - risponde alla domanda: come devo farlo?
        - riguarda il ways of working

    - **Classificazione <u>secondo la verificabilità</u>:**
      - per poter fare **validazione** (← glossario)

      - **requisiti funzionali:**
        - **descrivono i servizi che il sistema deve fornire**
          - esempio:
            - l'utente può aggiungere un asset
        - **accertiamo il soddisfacimento dei requisiti funzionali attraverso:**
          - **test**
          - **dimostrazione formale**
          - **revisione**

- **requisiti prestazionali:**
  - **definiscono quanto bene il sistema sta svolgendo determinate funzioni in certe particolari condizioni**
    - esempio:
      - velocità di risposta
      - tempo di esecuzione
      - capacità di memoria
  - **accertiamo il soddisfacimento dei requisiti prestazionali attraverso:**
    - **misurazione**

- **requisiti qualitativi:**
  - **servono a definire la qualità del prodotto**
    - esempio:
      - deve essere fornito un manuale utente
      - la codifica del prodotto rispetta le norme e le metriche indicate nei riferimenti normativi
  - **accertiamo il soddisfacimento dei requisiti qualitativi attraverso:**
    - **verifica ad hoc**

- **requisiti di vincolo:**
  - **sono imposti dal cliente o dall'ambiente operativo in cui funzionerà il sistema**
    - esempio:
      - l'applicazione deve funzionare su tablet
      - L'applicazione deve utilizzare il linguaggio JavaScript
  - **tipologie:**
    - **realizzativo**
    - **normativo**
    - **contrattuale**
  - **accertiamo il soddisfacimento dei requisiti di vincolo attraverso:**
    - **revisione**

- **Classificazione <u>secondo la diversa utilità strategica</u>:**
  - servono a negoziare i requisiti
    - quando li negozio?
      - non li negozio una volta per sempre
      - **li negozio durante tutto il progetto**
    - "spazio elastico di negoziato"
      - non ci congeliamo in un impegno che eventualmente più avanti vorremmo negoziare

  - **obbligatori:**
    - **irrinunciabili** per qualsiasi stakeholder
    - obbligo non negoziabile
    - shall!

  - **desiderabili:**
    - **danno un valore aggiunto di interesse**
      - se ci fossero sarebbe meglio
      - di interesse = a valore economico
    - non strettamente necessari
    - should!

  - **opzionali:**
    - **relativamente utili oppure contrattabili in seguito**
    - se l'utente li vuole allora deve pagare
      - non sono necessari
    - may!

- **Piano di Qualifica (V&V):**
  - **è un documento di strategia, legato all'obiettivo di efficacia**
    - **Qualifica = Verifica & Validazione**
    - **indica se stiamo soddisfacendo i requisiti**
  - **per garantire il rispetto dei requisiti**, entra in gioco il Piano di Qualifica
    - il Piano di Qualifica **nasce assieme ai requisiti**
  - **dice come faremo le cose nel ramo ascendente del modello a V**
    - quando siamo **in Analisi dice verificheremo se i requisiti sono stati coperti**
    - quando siamo **in Progettazione Logica dice come faremo codifica, secondo quale struttura**
    - quando siamo **in Progettazione di Dettaglio dirà come verificheremo il codice prodotto**
  - **dice come faremo verifica**
  - **dice, successivamente, con quale esito è stata fatta la verifica**
  - **definisce metodi, tecniche e procedure per fare validazione**

Analisi dei requisiti:

- **e' un'attività del processo di sviluppo**
  - non stiamo realizzando niente, serve a capire il problema
  - interagiamo con il cliente ed eventualmente anche con gli utenti
    - molte volte non coincidono queste due figure

- **Come fare analisi dei requisiti:**
  - l'analisi dei requisiti è un'**attività** che si ripete più volte
  - occorre fissare, scrivere i concetti sulla quale siamo d'accordo
    - la scrittura di testo scritto può essere ambigua
      - si fanno dunque dei disegni
        - **diagrammi dei casi d'uso**
          - raccolgono requisiti di chiunque userà parti del sistema
          - precedono i requisiti
            - sono una fonte dei requisiti
          - sono strumenti con i quali inizio un ragionamento con un approccio **top-down**
            - identificando il sistema e gli utenti, esterni ad esso
  - successivamente discuto quanto ho scritto…
  - ...poi torno indietro…
  - poi ridiscuto…
  - ...fino a che non ho soddisfatto gli stakeholders, e siamo entrambi d'accordo
  - ma non ho finito:
    - c'è distanza tra un requisito utente e un requisito relativo a come si fa una data cosa
      - voglio la crostata, intera
        - ma il cliente non vede gli ingredienti che servono per farla
          - compra marmellata
          - compra zucchero
          - ecc…
    - dobbiamo quindi continuare (noi, fornitori) a ricercare requisiti
      - fino ad arrivare ad un documento che definisce il problema, in maniera completa
        - il progettista prenderà in input questo problema

- le attività che faccio nell'analisi dei requisiti vanno **documentate**
  - devono produrre cose **leggibili** e **tracciabili**
  - sarà un documento certamente **versionato**

- **Attività dell'analisi dei requisiti:**
  - il processo di ingegneria dei requisiti **raggruppa quattro attività:**
    - **Studio di fattibilità**
    - **Acquisizione e analisi dei requisiti**
    - **Specifica dei requisiti**
      - cioè formalizzare i requisiti e riportarli in un documento
    - **Validazione dei requisiti**

- **Studio di fattibilità:**
  - **fa parte del processo di supporto "Documentazione"**
  - è uno **studio breve e chiaro che consiste nel valutare rischi, costi e benefici legati al sistema da sviluppare**
    - rischi: troppo complicato, troppo costoso, rapporto difficile con il cliente, ecc...
  - **deve descrivere in modo chiaro gli obiettivi del progetto e valutare approcci alternativi**
  - **è un' attività che produce un documento: lo studio di fattibilità**
    - serve a vedere se vale la pena ad ingaggiarsi per un dato progetto o meno
    - deve darci informazioni quantitative
      - identificando pro e contro

- **Acquisizione e analisi dei requisiti:**
  - dopo aver compiuto uno studio di fattibilità
  - **gli analisti devono lavorare assieme ai clienti e utenti per individuare il dominio di applicazione e i requisiti del sistema**

  - **Come si svolge:**

    - **Studio dei bisogni e delle fonti:**
      - si cerca di individuare un insieme, non strutturato, di requisiti.
      - **per fare ciò, si può:**
        - **studiare il ways of working**
          - è fonte di requisiti (non li valido, li verifico)
        - **esplodere i bisogni** in più sotto-bisogni
        - **considerare anche i bisogni non espressi**, non detti
        - **studiare il dominio**
        - **interrogare gli stakeholder**
          - deve produrre un documento scritto
        - **fare brainstorming**
          - deve avere un tempo finito, stabilito a priori, sapendo così quanto mi costerà
            - **quantificabile**
          - occorrente:
            - facilitatore : mantiene la discussione entro gli obiettivi e i tempi, senza partecipare attivamente
            - una persona che scriva la sintesi, registrando i punti salienti
            - persone che discutono
        - **studiare prototipi del sistema**
          - es: mockup per fissare delle aspettative

- **Classificazione e organizzazione dei requisiti**
  - l'insieme dei requisiti viene strutturato

- **modellazione concettuale del sistema (visione Use Case):**
  - ad esempio tramite diagrammi dei casi d'uso
  - divido il problema in parti e vedo come queste parti sono legate
    - sono più facili da controllare, da capire

- **assegnazione dei requisiti a parti distinte del sistema**

- **negoziazione con il committente e con i sotto-fornitori**
  - essendoci diversi stakeholders, molto spesso i requisiti sono in conflitto
  - viene data una priorità ai requisiti, negoziando su quelli incompatibili

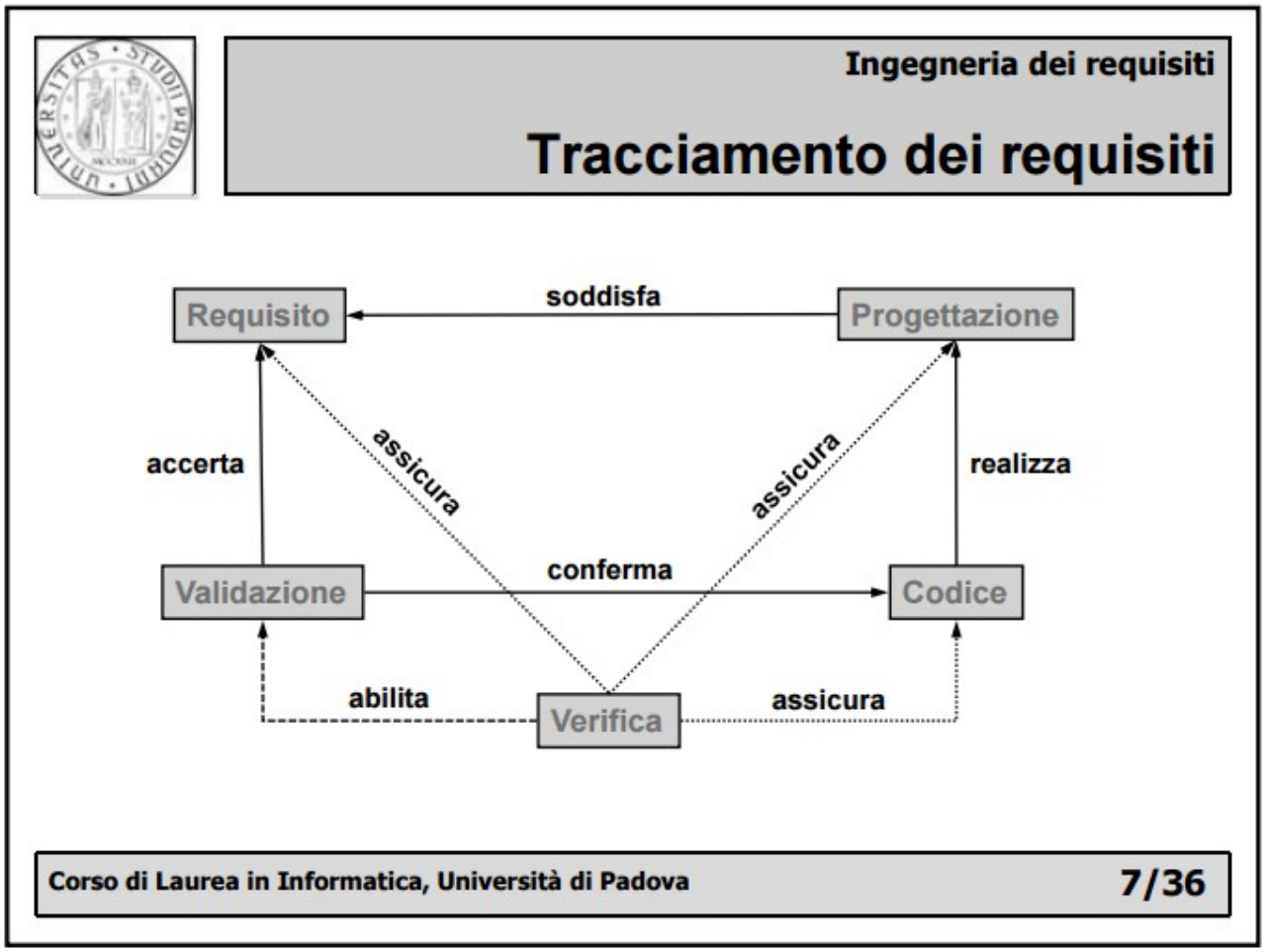
- **Specifica dei requisiti:**
  - **cioè formalizzare i requisiti e riportarli in un documento**
  - **usando** un linguaggio formale e/o grafico (**diagrammi UC**)
  - **i requisiti devono essere ordinati per priorità, classificati e identificati univocamente**
  - **lo standard IEEE 830 raccomanda che la specifica dei requisiti sia:**
    - **priva di ambiguità**
    - **corretta**
    - **completa**
    - **verificabile**
    - **coerente**
    - **modificabile**
      - se ho un sistema sciocco e manuale, le modifiche lascerebbero “buchi” tra gli identificatori
        - ad esempio non va bene usare una tabella excel
          - ho i requisiti raggruppati in sottogruppi, ognuno con un proprio identificatore. Se ne cancello uno in mezzo, rimane un buco tra gli indici degli identificatori.
        - => usare una base di dati
          - **l’identificazione deve quindi essere automatizzata**
    - **tracciabile**
    - **ordinata per rilevanza dei requisiti**

- **lo standard (IEEE 830) raccomanda inoltre che un documento di Analisi dei Requisiti contenga:**
  - **un'introduzione:**
    - scopo del documento
    - scopo del prodotto
    - glossario
      - voci in ordine alfabetico
      - raccoglie e definisce i termini chiave del dominio
      - è utile se:
        - mi aiuta nella lettura
        - non è intrusivo
        - risolve l'overloading dei termini
      - dev'essere automatizzato (G a pedice)
    - riferimenti
    - struttura del documento
  - **una descrizione generale:**
    - prospettive del prodotto
    - funzioni del prodotto
    - caratteristiche generali
    - vincoli
    - assunzioni e dipendenze

- **Validazione dei requisiti:**
  - Validare i requisiti **vuol dire controllare che essi definiscano effettivamente il sistema che il cliente richiede.**
  - **A partire dal documento generato nella Specifica dei Requisiti, ci assicuriamo che siano:**
    - **non ambigui**
    - **necessari:**
      - nessuna caratteristica superflua
      - tutti i requisiti in AR soddisfano un particolare bisogno
      - **come facciamo ad assicurarlo?**
        - attraverso il **tracciamento** (← glossario)
    - **sufficienti:**
      - nessun bisogno trascurato
      - tutti i bisogni rilevati nelle fonti, sono requisiti in AR
      - **come facciamo ad assicurarlo?**
        - attraverso il **tracciamento** (← glossario)
    - **coerenti**
    - **realistici**
      - implementabili con le tecnologie a disposizione
    - **verificabili**
      - si dovrà essere in grado di dimostrare che il sistema soddisfa i requisiti

- **Documenti coinvolti:**
  - aiutano a svolgere l’analisi dei requisiti
  - **Capitolato d’appalto:**
    - prodotto dal cliente
    - è una specifica dei bisogni
  - **Studio di fattibilità:**
    - il fornitore(noi) decide se gli interessa
      - se ci interessa, produciamo il prossimo documento
  - **Analisi dei requisiti:**
    - il fornitore dice al cliente “ho capito questo”
    - viene presentato al cliente

Vediamo la figura:



- **4 perni di un progetto:**
  - **Requisiti:**
    - un **progetto** parte solo se ci sono i requisiti
  - **Progettazione:**
    - dopo i requisiti c’è la progettazione
    - **idea di una soluzione che soddisfa i requisiti:**
      - **tutti i requisiti!**
      - **come faccio a dirlo?**
        - NON applico la validazione (← glossario)
        - **applico la verifica** (← glossario)
          - la progettazione si porta uno zaino che dice “io sono così perché soddisfo i requisiti”
          - **controllo quindi che ci sia il tracciamento** (← glossario).
            - **controllo che il tracciamento sia esaustivo, ovvero controllo che i requisiti siano effettivamente soddisfatti**

- **Codice:**
  - è buono se e solo se **realizza la progettazione:**
    - realizza esattamente ciò che ho progettato
      - **soddisfacendo così i requisiti**
  - **applico la verifica** (← glossario)
    - non basta che compili!
    - devo dimostrare che il codice realizza la progettazione

- **Validazione:**
  - **guarda il prodotto e smarca i requisiti**
    - soddisfatto? Si
    - soddisfatto? Si
    - soddisfatto? Si

- **Sul disegno, “Verifica” ha le linee tratteggiate in quanto è presente ovunque e non ha output tangibili**
  - **la verifica affianca inoltre tutti gli stati del ciclo di vita, quindi non corrisponde ad attività che fanno avanzare i processi**

Fine lezione T7

Lezione T8 – Data: 21/10/2016 – Titolo: Analisi dei requisiti_parte2

- **Struttura del documento di Analisi dei Requisiti nel nostro progetto:**
  - **Introduzione:**
    - scopo e struttura
  - **Descrizione generale:**
    - Contesto d'uso del prodotto
      - ambiente in cui il prodotto è usabile
    - Funzioni del prodotto
    - Caratteristiche degli utenti
    - Vincoli generali
      - L'applicazione dovrà essere integrabile nel sistema…
    - Assunzione e dipendenze
      - Per il corretto funzionamento dell'applicazione sarà necessario l'utilizzo di Google Chrome…
  - **Casi d'uso:**
    - Classificazione dei casi d'uso
      - UC[Codice padre].[Codice identificativo]
    - Attori
    - [...Elenco degli UC…]
  - **Requisiti:**
    - Classificazione dei requisiti
      - R[Importanza][Tipologia][Codice]
        - Importanza:
          - Obbligatori
          - Desiderabili
          - Facoltativi
        - Tipologia
          - Funzionale
          - Qualità
          - Prestazionale
          - Vincolo
    - Fonti
      - capitolato
      - verbale interno
      - verbale esterno
      - caso d'uso
      - interno
        - = identificato dagli analisti
    - Requisiti funzionali
    - Requisiti prestazionali
    - Requisiti qualitativi
    - Requisiti di vincolo

- **Tracciamento dei requisiti**
  - Tracciamento requisiti-fonti
  - Tracciamento fonti-requisiti

Fine set slide #6 - “Analisi dei requisiti”

Fine lezione T8

Lezione T9 – Data: 27/10/2016 – Titolo: **Progettazione**

Progettazione:

- Premessa:
  - **la risoluzione di un problema attraversa due fasi:**
    - **fase analitica:**
      - **il problema viene decomposto** e approfondito nel dettaglio per capire di quali parti è formato
        - espande il problema: da una formulazione semplice ad una formulazione ricca
      - **approccio top-down**
      - **risponde alla domanda: qual è il problema?**
        - corrisponde grossomodo all'attività dell'**analisi dei requisiti**
        - attività fondamentalmente investigativa
    - **fase sintetica:**
      - **si ricompongono i pezzi trovati al passo precedente e si sintetizza una soluzione per il problema**
      - **approccio bottom-up**
      - **risponde alla domanda: come si risolve il problema?**
        - corrisponde alla **progettazione**
          - **va documentata in modo non ambiguo**
        - la soluzione deve trovare il giusto equilibrio da efficacia ed efficienza (← glossario)
      - **costruisce l'architettura**
        - costruire il prodotto in modo tale da garantire il soddisfacimento delle proprietà identificate

- **Due tipologie di approcci che posso applicare alla progettazione (due spinte, due modi di effettuare breakdown/decomposizione):**
  - **approccio funzionale:**
    - Studio di fattibilità
    - Analisi dei requisiti
      - uso prevalente di linguaggio naturale
    - Specifica Tecnica
    - **Progettazione top-down (↓)**
      - ottengo **elevata coesione**
      - **basso grado di accoppiamento**
    - Programmazione procedurale

  - **approccio Object-Oriented (O.O)**
    - Studio di fattibilità
    - Analisi orientata agli oggetti (O.O)
      - uso prevalente di formalismi grafici (diagrammi “use case”)
      - continuità logica con la progettazione mediante prima identificazione delle classi
    - **Progettazione bottom-up (↑)**
      - prendo elementi base e li metto assieme
      - **riuso di componenti** prefabbricati
      - **tre modelli:**
        - **modello statico:**
          - **identifico classi e oggetti con attributi e associazioni, tramite:**
            - **aggregazione:** A è una parte di B
            - **generalizzazione:** C è un tipo di D
            - **utilizzo dell’ereditarietà come strumento di organizzazione, semplificazione e riuso**
        - **modello dinamico:**
          - guado al comportamento interno e interazioni tra componenti
        - **modello funzionale:**
          - guado ai tipi dei valori in ingresso/uscita e flusso dei dati
    - Programmazione OO

- **2 modi per raggiungere la correttezza:**
  - **trial and error:**
    - la mosca che vuole uscire e continua a sbattere contro alla finestra
    - provo finché non riesco
  - **raggiungere la correttezza per costruzione:**
    - fare cose che ci portano ad essere sicuri che il risultato è quello che volevamo
    - la progettazione precede la produzione

- **Perchè progettare? per:**
  - **dominare la complessità del problema**
    - **approccio divide-et-impera**
      - **breakdown**
      - rompiamo le parti, mantenendole però connesse così come sono e con un ruolo chiaro
  - **poter procedere in parallelo con il lavoro**
  - garantire efficacia ed efficienza (← glossario)

- **La progettazione costruisce l'architettura:**
  - la scelta di una buona architettura facilita il successo
  - **la progettazione è la definizione dell'architettura, delle componenti e delle interfacce e delle altre caratteristiche di un sistema o componente**

  - **Una definizione di architettura: in cosa consiste:**
    - **l'architettura di un software è, tipicamente, un insieme di moduli che si raggruppano in unità, che a loro volta si raggruppano in componenti, che vanno a formare un sistema. (SCUM)**
      - **un modulo rappresenta un carico di lavoro realizzabile dal singolo programmatore**

    - **fonte: ISO/IEC/IEEE 42010**
    - **Consiste nei seguenti passi:**
      - **Decomposizione del sistema in componenti**
      - **Organizzazione di tali componenti**
      - **Individuazione delle interfacce necessarie all'interazione tra le componenti tra loro e con l'ambiente**
      - **Definisco le regole di composizione delle componenti**

    - **Decomposizione del sistema in componenti:**
      - **L'architettura ha 2 stadi:**
        - **Progettazione logica/architetturale:**
          - **progettazione ad alto livello**
          - **riconosciamo le parti separabili, cercando di comprenderne la logica**
          - **nel Piano di Qualifica, la progettazione architetturale è verificata dai test di integrazione**
          - esempio:
            - come ci immaginiamo che un'applicazione (WhatsApp) sia fatta:
              - front-end:
                - è l'interfaccia utente
              - back-end:
                - è la base di dati
              - in mezzo c'è lo strato logico:
                - dice come le due parti interagiscono tra loro, e come l'informazione viaggia

        - **Progettazione di dettaglio:**
          - **descrive il comportamento delle componenti identificate nella progettazione logica**
          - **nel Piano di Qualifica, la progettazione di dettaglio è verificata dai test di unità**
          - **fino a che livello continuo a fare breakdown?**
            - non c'è una regola
            - **due criteri:**
              1. **continuo fino a che governare un oggetto più piccolo è più costoso/complesso rispetto alla situazione attuale**
              2. **devo poter assegnare le attività ad una singola persona**

- **L’organizzazione di tali componenti:**
  - definizione dei ruoli, responsabilità, interazioni
    - chi fa cosa e come

- **Le interfacce necessarie all’interazione tra le componenti tra loro e con l’ambiente:**
  - poiché le parti in cui abbiamo diviso il tutto devono collaborare per fare il loro compito complessivo
  - esempio di interfaccia:
    - TCP/IP
      - un protocollo è un accordo di interfaccia
        - se tu mi dici una cosa che io mi aspetto, risponderò in questo modo
        - le interfacce definiscono il modo in cui le parti(chiamate componenti) comunicano tra loro

- **Regole di composizione delle componenti:**
  - decidere come le cose stanno assieme
    - regole, criteri, limiti, vincoli
      - anche a fini di manutenibilità
    - esempio:
      - TCP/IP
        - è una pila
          - si parla solo a livelli adiacenti
            - si parla con quelli di sopra ricevendo un comando
            - si parla con quelli di sotto mandando un comando

- **Criteri guida per costruire una buona architettura:**
  - **Adottare stili architetturali:**
    - **lo stile è qualcosa che ha elementi riconoscibili e ricorrenti**
      - ricorrenti = pattern (← glossario)
        - **le architetture hanno lo stile se attuano dei pattern** (← glossario)
    - **il progettista applica, non inventa stili architetturali**
  - le *scelte architetturali* determinano l’organizzazione dell’informazione e l’interazione tra le parti

- **Qualità di una buona architettura:**
  - **devono essere tutte misurabili**
    - **non vogliamo raggiungere la correttezza tramite correzioni**

  - **Sufficienza:**
    - **è capace di soddisfare tutti i requisiti**
      - **dev'essere quindi possibile un tracciamento dei requisiti**

  - **Comprensibilità:**
    - dev'essere descritta in modo comprensibile, capibile dagli stakeholders

  - **Modularità:**
    - fatta di parti chiare e ben distinte
      - ciascuna con la sua funzione
      - senza sovrapposizioni nei compiti

  - **Robustezza:**
    - non vogliamo che crolli tutto se c'è una parte che non fa il suo compito correttamente
      - un singolo malfunzionamento potrebbe bloccare l'intero sistema
    - è capace di sopportare input diversi (giusti, sbagliati, tanti, pochi) dall'utente e dall'ambiente

  - **Flessibilità:**
    - permette modifiche a costo contenuto
      - al variare dei requisiti
      - a seguito di manutenzioni

  - **Riusabilità:**
    - le sue parti possono essere utilmente impiegate in altre applicazioni
    - riuso (← glossario)
      - problemi:
        - progettare per riuso è più difficile
          - bisogna anticipare bisogni futuri
            - facilmente adattabile al nuovo compito
        - progettare con riuso non è immediato
          - bisogna minimizzare le modifiche alle componenti riusati per non perderne il valore

  - **Efficienza:**
    - (← glossario)

  - **Affidabilità:**
    - è altamente probabile che svolga bene il suo compito

- **Disponibilità:**
  - necessita di poco o nullo tempo di manutenzione fuori linea
    - non tutto il sistema deve essere interrotto se qualche sua parte è sotto intervento
      - != UNIWEB

- **Sicurezza rispetto a malfunzionamenti (safety):**
  - il sistema dispone di un sufficiente grado di ridondanza

- **Sicurezza rispetto a intrusioni (security):**
  - dati e funzioni non sono vulnerabili a intrusioni

- **Semplicità:**
  - l'architettura non dev'essere mai inutilmente complessa
    - ogni parte contiene solo il necessario e niente di superfluo
    - le cose semplici sono infatti:
      - più comprensibili
      - più spiegabili
      - più manutenibili
      - più verificabili
  - **Principio del rasoio di Occam:**
    - "le entità usate per spiegare un fenomeno non devono essere moltiplicate senza necessità"
      - dico cose per spiegare, ma quelle che dico devono essere necessarie e non superflue
    - adottato da Isaac Newton nella fisica
      - "quando hai due soluzioni equivalenti rispetto ai risultati scegli la più semplice"
    - e poi anche da Albert Einstein
      - Everything should be made as simple as possible, but not simpler
        - tutto dovrebbe essere ridotto al minimo indispensabile, ma non ancora di più

- **Incapsulamento:**
  - nasconde i dettagli implementativi
    - le componenti sono "black box"
  - benefici:
    - l'esterno non può fare assunzioni sull'interno
    - cresce la manutenibilità
    - diminuendo le dipendenze aumentano le opportunità di riuso

- **Coesione:**
  - **le parti sono raccolte in gruppi che concorrono agli stessi obiettivi e sono strettamente e necessariamente coinvolte**
  - **è un indicatore di necessità dei membri che ci sono dentro**
    - **se ne manca uno non sono sufficienti**
    - **se ce n'è uno in più, è superfluo**
  - **proprietà interna di singole componenti**
    - **funzionalità "vicine" devono stare nella stessa componente**
  - **va massimizzata**
  - **l'orientamento ad oggetti aiuta molto a raggiungere la coesione : SOLID**
    - **S = Singe responsability principle**
      - **ogni classe dovrebbe avere una sola responsabilità, interamente incapsulata al suo interno**
    - **O = Open/close principle**
      - **un'unità dovrebbe essere aperta alle estensioni, ma chiusa alle modifiche**
    - **L = Liskov substitution principle**
      - **gli oggetti dovrebbero poter essere sostituiti con dei loro sottotipi**
    - **I = Interface segregation principle**
      - **è preferibile avere più interfacce specifiche, anziché una generica**
    - **D = Dependency Inversion principle**
      - **una classe dovrebbe dipendere dalle astrazioni, non dalle classi concrete**
  - Benefici:
    - maggiore manutenibilità e riusabilità
    - minore interdipendenza fra componenti
    - maggiore comprensione dell'architettura del sistema
  - **Buone forme di coesione:**

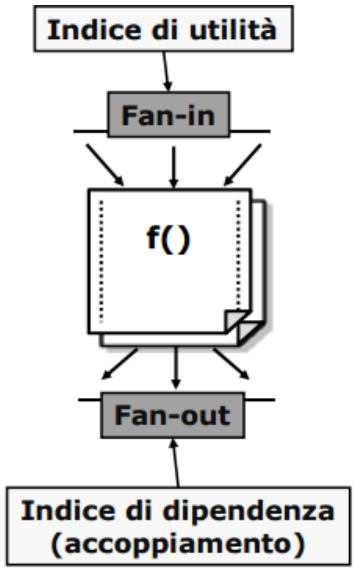
    - **Funzionale:**
      - quando le parti concorrono al <u>medesimo specifico compito</u>
      - **ci sono due modi per ottenerla:**
        - **top-down:**
          - servono più elementi per svolgere un certo compito
          - ottengo coesione funzionale per decomposizione
        - **bottom-up:**
          - cosa devo fare nel piccolo?
          - Se penso che le azioni siano funzionalmente correlate, le metto assieme

    - **Sequenziale:**
      - quando alcune azioni sono <u>"vicine" ad altre per ordine di esecuzione</u> e dunque conviene tenerle insieme

    - **Informativa:**
      - quando le parti agiscono sulla <u>stessa unità di informazione</u>

- **Basso accoppiamento:**
  - **parti distinte dipendono poco o niente le une dalle altre**
  - **va minimizzato**
  - **un sistema è un insieme organizzato**
    - dunque **ha necessariamente un po' di accoppiamento**
      - la buona progettazione lo tiene basso
  - **descrive dipendenze indesiderabili**
    - **vogliamo rimuoverle**
      - **facendo ad esempio parlare le componenti tramite interfacce**
    - **esempi** di cattiva interdipendenza:
      - **fare assunzioni dall'esterno su come le parti facciano il loro mestiere all'interno** (per variabili, locazioni, tipi)
        - es: lavoro su una coda direttamente
          - anziché usare ad esempio un'interfaccia
      - **imporre vincoli dall'esterno sull'interno** (per ordine di azioni, uso di certi dati, formati, valori)
        - esempio: tipo "data" e millenium bug
      - **condividendo frammenti delle stesse risorse** (strutture dati)

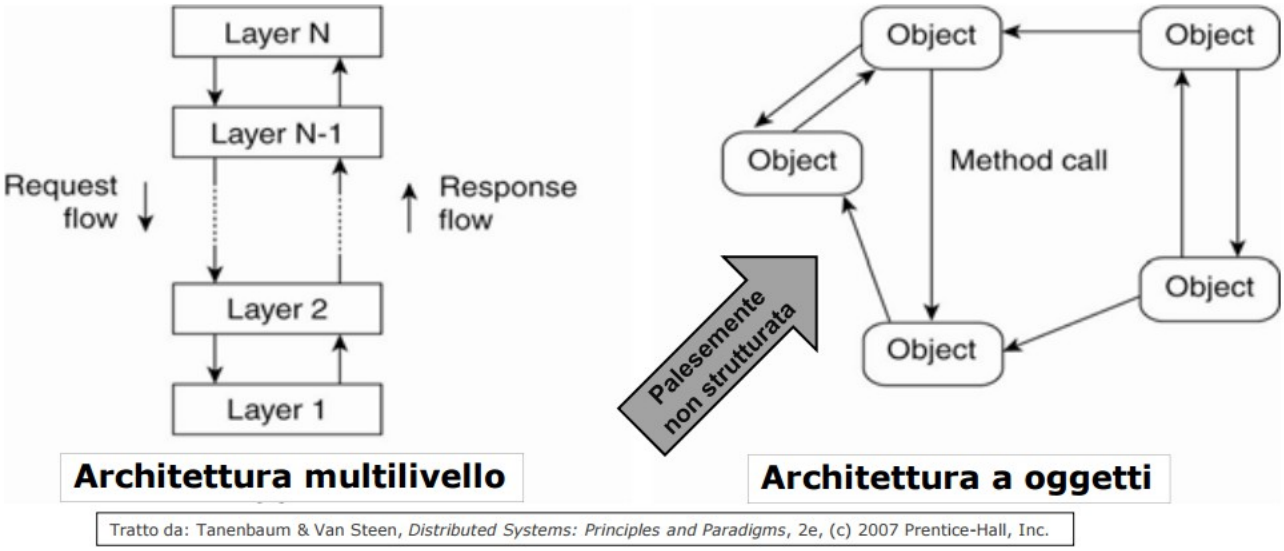
  - **L'accoppiamento è misurabile interpretando le componenti di un sistema come i nodi di un grafo orientato dove gli archi sono dipendenze** di una componente nei confronti di un'altra
    - **il numero di archi entranti in una componente (fan-in) è indice di utilità**
      - **va massimizzato**
    - **il numero di archi uscenti (fan-out) è indice di dipendenza da altre componenti**
      - **va minimizzato**
    - **il massimo accoppiamento è dato da M x M**



- **Design pattern e stili architetturali:**
  - **Soluzione progettuale generale a un problema ricorrente.**
  - **Definisce una funzionalità lasciando gradi di libertà d'uso**
    - ha corrispondenza precisa nel codice sorgente
  - **Soluzioni progettuali ad alto livello, a livello di sistema => pattern architetturali**
  - **Soluzioni progettuali a basso livello => design pattern**
  - Concetto promosso da C. Alexander (un vero architetto)
    - **l'architettura fa emergere degli stili**
    - **un'architettura ha stile se applica dei pattern**

- **Pattern architetturali:**
  - **esempi di pattern architetturali:**
    - **architettura "three-tier":**
      - **il sistema è fatto di 3 livelli:**
        - **interfaccia utente in alto:**
          - GUI
        - **logica in mezzo:**
          - business logic
        - **persistenza in basso:**
          - strato dell'organizzazione dei dati
            - database
          - ogni applicazione utile ha persistenza

    - **variante multi-livello:**
      - pila OSI
      - TCP/IP



Tratto da: Tanenbaum & Van Steen, *Distributed Systems: Principles and Paradigms*, 2e, (c) 2007 Prentice-Hall, Inc.

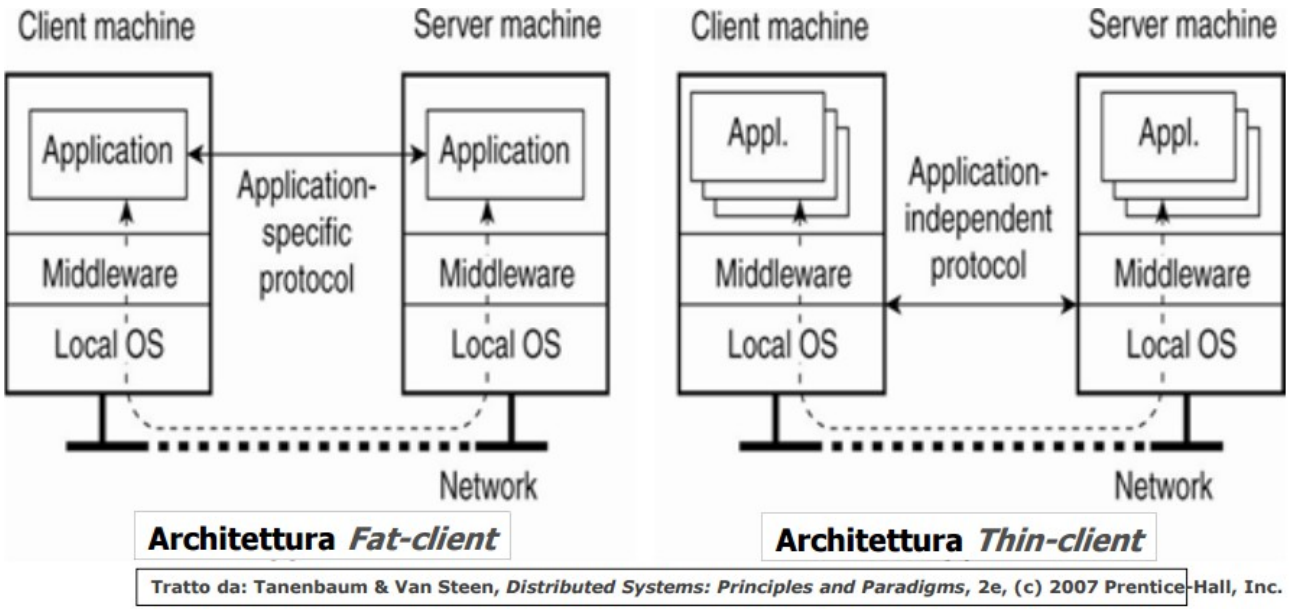
      - Nell'esempio, nell'architettura a oggetti, chiunque parla con chiunque
        - palesemente non strutturata

    - **architettura produttore-consumatore:**
      - collaborazione a pipeline
      - consiste in un produttore e un consumatore, e l'utilizzo di una coda

- **architettura client-server:**
  - nella figura in basso:
    - il sistema è strutturato a livelli
    - ho due tipologie di clienti
      - la differenza sta nel protocollo
        - protocollo = accordo tra parti (chi fa cosa e come)

      - **fat-client:**
        - **minor carico sul server**
        - **scarsa portabilità**
          - **se il server parlasse con altri client, diversi, allora il server dovrebbe parlare più protocolli**
            - **<u>=> accoppiamento!</u>**
        - **lo strato applicazione comunica con lo strato applicazione**
        - il client possiede la logica e capisce il protocollo con il quale deve parlare

      - **thin-client:**
        - **maggior carico di comunicazione**
        - **buona portabilità**



- **architettura “peer-to-peer”:**
  - interconnessione di scambio senza server intermedio

Fine lezione T9

Lezione T10 – Data: 28/10/2016 – Titolo: Progettazione_parte2

- **Progettazione di dettaglio:**
  - **si ricorre ai design pattern, che si dividono in 3 famiglie:**

    - **design pattern creazionali:**
      - **cercano di rendere un sistema indipendente dall'implementazione concreta delle sue componenti**

    - **design pattern strutturali:**
      - **affrontano problemi riguardanti la composizione di classi e oggetti**

    - **design pattern comportamentali:**
      - **si occupano del comportamento degli oggetti e delle collaborazioni tra essi**

  - **obiettivi:**
    - **assegnare unità a componenti:**
      - per organizzare il lavoro di programmazione
      - per assicurare congruenza con l'architettura di sistema
    - **produrre la documentazione necessaria:**
      - perché la programmazione possa procedere in modo certo e disciplinato
      - **tracciamento dei requisiti alle unità**
    - **definire i test di unità**

- **Documentazione:**
  - **IEEE 1016**: Software Design Document
    - Introduzione
      - come nel documento AR
    - Riferimenti normativi e informativi
    - Descrizione della decomposizione architetturale
      - moduli, processi, dati
    - Descrizione delle dipendenze, tra:
      - moduli, processi, dati
    - Descrizione della progettazione di dettaglio

- **Stati di progresso per SEMAT:**
  - **definisce possibili milestone** (← glossario)**, associati alla progettazione:**
    - **avere una sola milestone per la progettazione è troppo poco**
      - determinerebbe successo o fallimento, esito binario
    - **Architecture selected:**
      - **“abbiamo capito che architettura dobbiamo usare”**
      - selezione delle tecnologie necessarie
    - **Demonstrable:**
      - **“ho un’architettura le cui caratteristiche essenziali possono essere dimostrate, argomentate, spiegate”**
    - **Usable:**
      - **“a questa data, la mia architettura può fare questo”**
    - **Ready:**
      - **“la mia architettura è pronta all’uso”**
      - la documentazione per l’utente è pronta
      - gli stakeholder hanno accettato il prodotto e vogliono che diventi operativo

Fine set slide #7 - “Progettazione software”

Fine lezione T10

Lezione T11 – Data: 03/11/2016 – Titolo: Documentazione

Flipped classroom: studio autonomo e presentazione in aula di strumenti di:

- analisi dei requisiti
- piano di qualifica
- norme di progetto

Documentazione:

- **Documentazione è tutto il materiale che documenta le attività di un progetto (e il loro prodotti)**

- **Perchè documentare:**
  - **in Ingegneria, l'obiettivo primario della documentazione di un prodotto è la replicabilità di tale prodotto**
  - **per i membri del progetto la documentazione rappresenta uno strumento di comunicazione**
  - **per i programmatori la documentazione segna il confine tra creatività e disciplina**
  - **per gli utenti del prodotto, alcuni documenti sono utili per capire come interfacciarsi al prodotto**
  - **per chi dovrà manutenere il prodotto, servono da repository di informazioni sul sistema**
  - **forniscono informazioni che aiutano la pianificazione dello sviluppo**
  - **aiutano ad assicurare la qualità, secondo metriche ben fondate**
  - **aiutano a fissare i requisiti, evitando dispersioni e inconsistenze**

- **Ciascun documento avrà un preambolo:**
  - **lista di distribuzione:**
    - dice chi l’ha redatto
    - chi l’ha modificato
    - i destinatari
  - **registro delle modifiche:**
    - i documenti evolvono, maturano
    - sottoposto a regole interne
    - prodotto da un sistema di versionamento
  - **scopo del documento**
  - **scopo del prodotto**
  - **riferimenti:**
    - riferimenti normativi
    - ...e altri riferimenti
  - **glossario:**
    - avrò un riferimento al glossario
      - **per una ragione principalmente di consistenza, il glossario sarà separato dal documento**

- **I principali documenti di progetto sono:**
  - **Analisi dei requisiti**
  - **Specifica Software**
    - descrizione ad alto livello del sistema
  - **Specifica Tecnica**
    - architettura logica
  - **Definizione di Prodotto**
    - architettura di dettaglio
  - **Manuale utente**

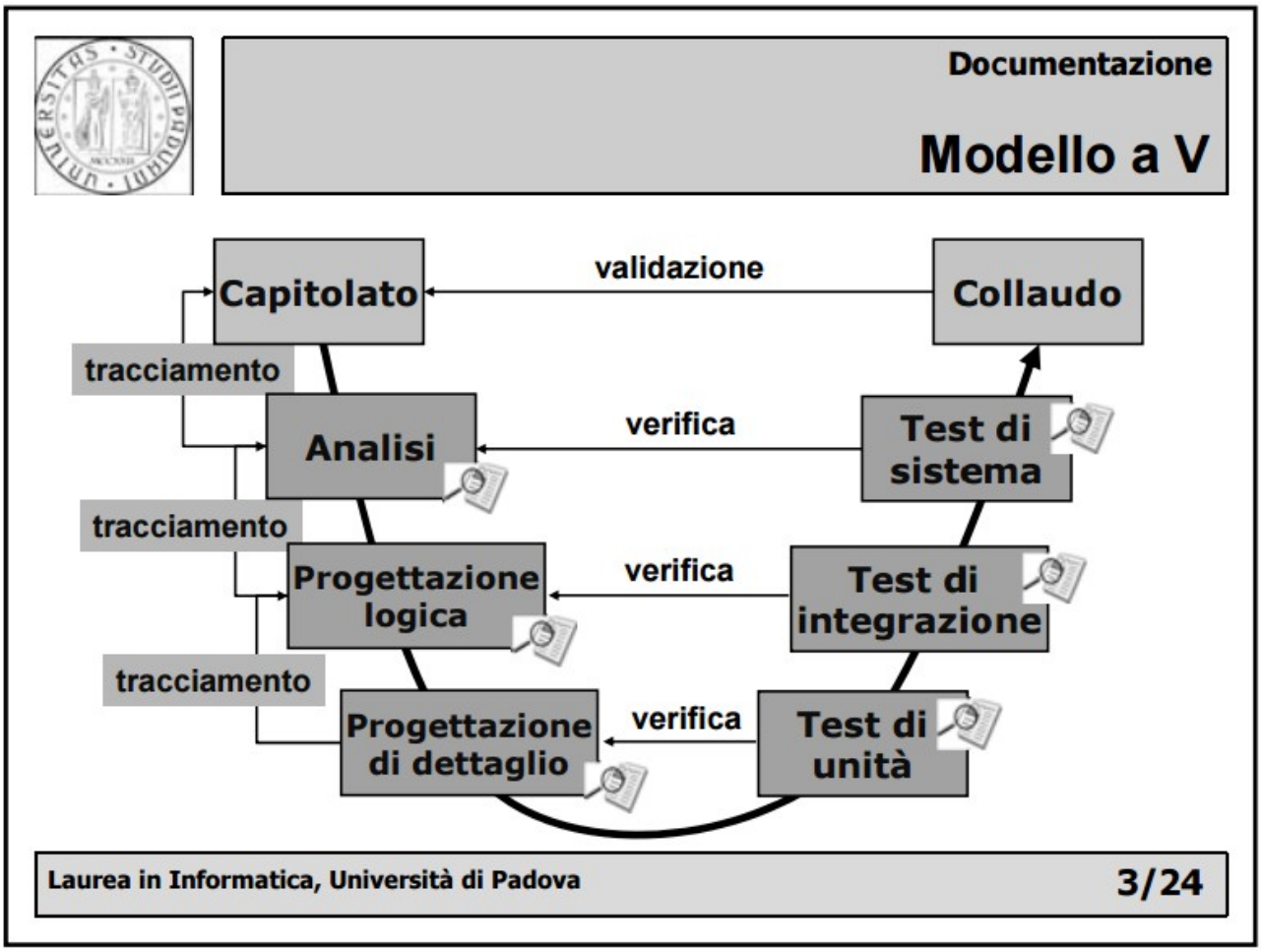
- **Specifica Software:**
  - **descrizione ad alto livello del sistema tramite rappresentazione gerarchica**
    - **usando notazioni e convenzioni coerenti**
      - **es: UML**
  - **modelli di specifica software:**
    - ogni architettura software ha molte “viste”
      - **modello statico:**
        - diagrammi di struttura
        - identifica le componenti principali e procede per decomposizione gerarchica
      - **modello dinamico:**
        - diagrammi di comportamento
        - illustra la struttura “a processi” del sistema
      - **modello delle interfacce:**
        - diagrammi di struttura
        - definisce le interfacce richieste da componenti del sistema
      - **modello delle relazioni:**
        - diagrammi di interazione
        - identifica il flusso dei dati tra componenti distinti in relazione tra loro
      - **modello di distribuzione:**
        - diagrammi di deployment
        - mostra l’associazione tra nodi fisici e componenti logiche

- **Specifica Tecnica (ST):**
  - **architettura logica → specifica tecnica**
    - **descrive l'architettura logica del sistema**
  - **prodotta a valle dell’analisi dei requisiti**
  - **mostra ciò che il sistema deve fare, senza però fissare i dettagli implementativi**
  - **descrive l'interfaccia di ogni componente del sistema**
    - **attraverso più livelli gerarchici di decomposizione**
    - **per ogni componente del sistema, viene specificato:**
      - **funzione svolta**
      - **tipo dei dati in ingresso**
      - **tipo dei dati in uscita**
      - **risorse logiche e fisiche necessarie per il suo funzionamento**

- **Definizione di Prodotto (DP):**
  - **architettura di dettaglio → definizione di prodotto**
    - **descrive l'architettura di dettaglio del sistema**
  - **procede dall’architettura logica**
  - **consente sviluppo parallelo dei componenti terminali**
  - **consente di stimare costo e tempi di realizzazione**
  - **decompone i componenti architetturali in moduli a grana più fine**
    - fino a che ogni modulo ha dimensione, coesione, complessità e accoppiamento appropriati per codifica in parallelo
  - **il DP deve fornire tutti i dettagli necessari alla codifica e la verifica di ciascun modulo**

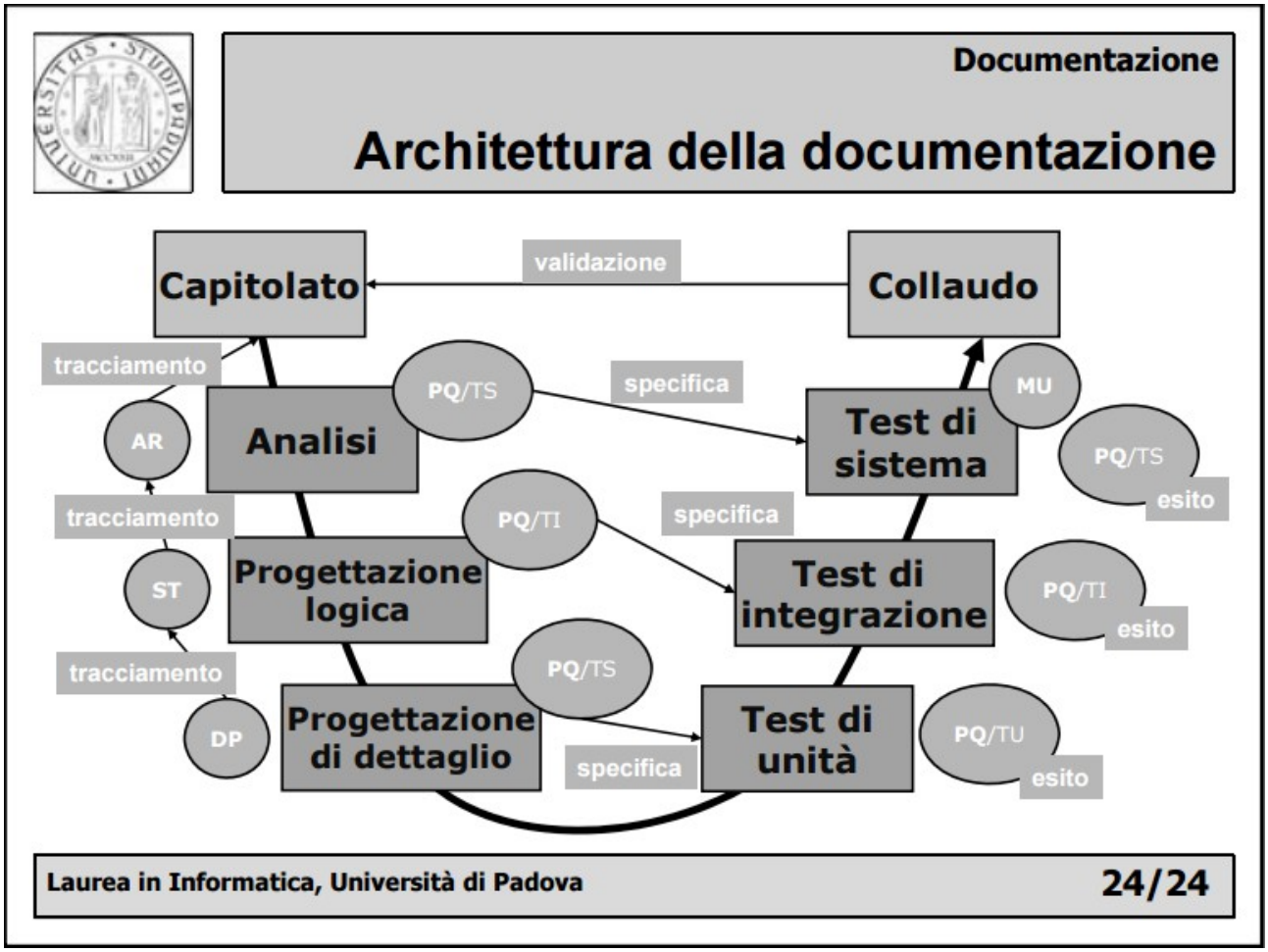
- **Analisi dei requisiti (AR):**
  - **nell'analisi dei requisiti, assume un ruolo di grande rilevanza il tracciamento** (← glossario) **dei requisiti**

- **Manuale utente (MU):**
  - **ha la funzione di spiegare all'utente come utilizzare il prodotto**
  - **nasce presto e cresce con il prodotto**
  - **deve avere:**
    - **frasi brevi**
    - **paragrafi focalizzati**
    - **liste**
    - **stile semplice**
    - **terminologia precisa**
    - **usare la forma attiva**
    - **… oltre che essere grammaticalmente corretto**
  - **può essere di vari tipi, a seconda del grado di esperienza dell'utente**
    - **manuale introduttivo**
    - **manuale completo**
  - **può avere diversi formati**
    - **cartaceo**
    - **PDF**



- **Modello a V per lo sviluppo del software:**
  - **Rappresenta il flusso ordinato dei prodotti per il lavoro**
  - **Ramo discendente:**
    - **il ramo discendente determina il modo in cui si farà verifica**
    - parto dal problema fissato dal committente (capitolato) e dal dominio
    - gli analisti studiano il problema e producono il documento AR
    - i progettisti prendono in input l'analisi dei requisiti e cercano di ricavarne una soluzione appropriata a quel problema, documentandola in due avanzamenti:
      - Progettazione Logica → ST
      - Progettazione di Dettaglio → DP
    - portano ad avere una soluzione
  - **Al vertice:**
    - c'è la **codifica**
      - viene codificato quanto precedentemente progettato
  - **Ramo ascendente:**
    - progressivamente si mettono assieme i pezzi che i programmatori hanno prodotto
      - non tutti insieme in una volta!
        - perché è la migliore prassi (best practice) per fare le cose in modo ordinato
      - verifichiamo ad ogni passo i pezzi di codice, fino ad arrivare ad un sistema che rappresenta l'implementazione di ciò che abbiamo immaginato essere la soluzione al problema
        - così possiamo arrivare al collaudo con il committente

- **Valutazione quantitativa del prodotto:**
  - **non serve misurare tutto indistintamente**
    - **meglio concentrarsi su quanto serve a specifiche necessità di miglioramento**
      - **secondo obiettivi strutturali e priorità fissate dalla direzione**
      - per produrre effetti permanenti nell'organizzazione
  - **alcune metriche di prodotto:**
    - **Dimensione**
    - **Struttura:**
      - flusso di controllo, flusso dei dati, annidamento, modularità e interazione
    - **Uso delle risorse:**
      - risorse fisiche e logiche (spazio di memoria, tempo di esecuzione)
      - risorse tecniche (strumenti)
      - risorse umane (tempo/persona)
    - **Qualità:**
      - **ISO/IEC 9126 Software product quality** (1999-2001)
        - **prevede una serie di normative e linee guida per descrivere un modello della qualità del software**
  - **trattamento dei dati di misurazione:**
    - **selezionare le misure di maggior uso potenziale secondo gli obiettivi prefissati, a costo contenuto di determinazione e proporzionato ai benefici attesi**
    - **i dati vanno valutati**



Fine set slide #8 - "Documentazione"

Fine lezione T11

Lezione T12 – Data: 17/11/2016 – Titolo: **Qualità di prodotto**

Qualità del software:

- **Definizione di qualità:**
  - **Secondo ISO, la qualità è l'insieme delle caratteristiche di un'entità che ne determinano la capacità di soddisfare esigenze espresse e implicite.**
  - **E' funzione del ways of working**
  - **E' la miglior misura che abbiamo della conformità**
    - vogliamo misurarla in modo oggettivo e non retrospettivo
    - **misuro l'efficacia e l'efficienza**

  - **Valutare la qualità serve a:**
    - **chi ha realizzato il prodotto (per migliorarlo)**
    - **chi lo usa (per avere garanzie)**
    - **chi valuta il prodotto (vogliamo valutazioni oggettive e non soggettive)**

  - **Qualità è un concetto ambivalente:**
    - **può riguardare la conformità ai requisiti**
      - visione intrinseca della qualità
    - **può riguardare la soddisfazione del cliente**
      - visione relativa
    - **può riguardare la misura del livello di qualità**
      - visione quantitativa

- **Sistema qualità:**
  - **ISO definisce il "sistema qualità" come la struttura organizzativa, le responsabilità, le procedure, i procedimenti e le risorse messe in atto per il perseguimento della qualità**
  - **Rappresenta un modo di lavorare (quindi norme e strumenti) che aiutano a produrre qualità**
  - **Viene documentato nel ways of working, ovvero nelle Norme di Progetto**
    - **sta alla base di un sistema qualità**

  - **Gestisce la qualità in 3 ambiti:**

    - **Pianificazione:**
      - date dalle **attività mirate a fissare gli obiettivi di qualità**, con i processi e le risorse necessari per conseguire tali obiettivi
      - è una premessa al controllo qualità

    - **Controllo:**
      - date dalle **attività per far sì che che il prodotto soddisfi i requisiti attesi**
      - **ha due facce:**
        - **retrospettiva:**
          - prima faccio e poi controllo
        - **preventiva:**
          - si chiama **Quality Assurance:**
            - **insieme di attività che attuo per garantire che il prodotto sia almeno tanto buono quanto lo volevo**
              - **attraverso verifiche**
                - **analisi statica e analisi dinamica sono parte della quality assurance**
              - **è un controllo che viene fatto sui processi con i quali si sviluppa il prodotto**
            - **serve a raccogliere tempestivamente evidenza oggettiva e di qualità**
            - **standard ISO / IEC 9126 come riferimento**
              - @qualità di prodotto
              - lezione T12 per vederne le caratteristiche

    - **Miglioramento continuo:**
      - **secondo lo schema PDCA**

- **Impatto degli standard sulla qualità:**
  - **sono delle raccolte di best practice** (← glossario)
  - **gli standard cercano di aumentare la qualità**
    - sono delle norme che emergono per il consenso mondiale
  - **il lato negativo degli standard:**
    - tipicamente le persone tendono a vedere gli standard come cose irrilevanti o bloccanti o fastidiose
    - l'attuazione di standard svincolata da controlli di efficacia sfocia in eccessi di burocrazia
      - il mio fare deve darmi implicitamente informazioni
    - **senza il supporto di strumenti informatici possono richiedere frustranti attività manuali**
      - i supporti informatici tolgono l'atto aggiuntivo

  - **sviluppi e integrazioni allo standard ISO/IEC 9126**
    - **ISO/IEC 9126:**
      - **definisce un modello della qualità del software**
    - **ISO/IEC 14598:**
      - **associa una metrica** (← glossario) **alle caratteristiche individuate dallo standard ISO/IEC 9126**
    - **ISO/IEC 25000:**
      - **riunisce 9126 e 14598, creando un progetto più specifico che prende il nome SquaRE**

- **Modelli concettuali della qualità:**
  - **per uniformare la percezione e la valutazione della qualità è bene che i committenti e i fornitori si accordino su un modello da seguire per la qualità**
  - **è fonte di requisiti**
    - è implicata a ogni dominio
  - **standard ISO/IEC 9126**
    - **la qualità di prodotto è fatta di 7 caratteristiche principali:**
      - suddivise in **31 sotto-caratteristiche**

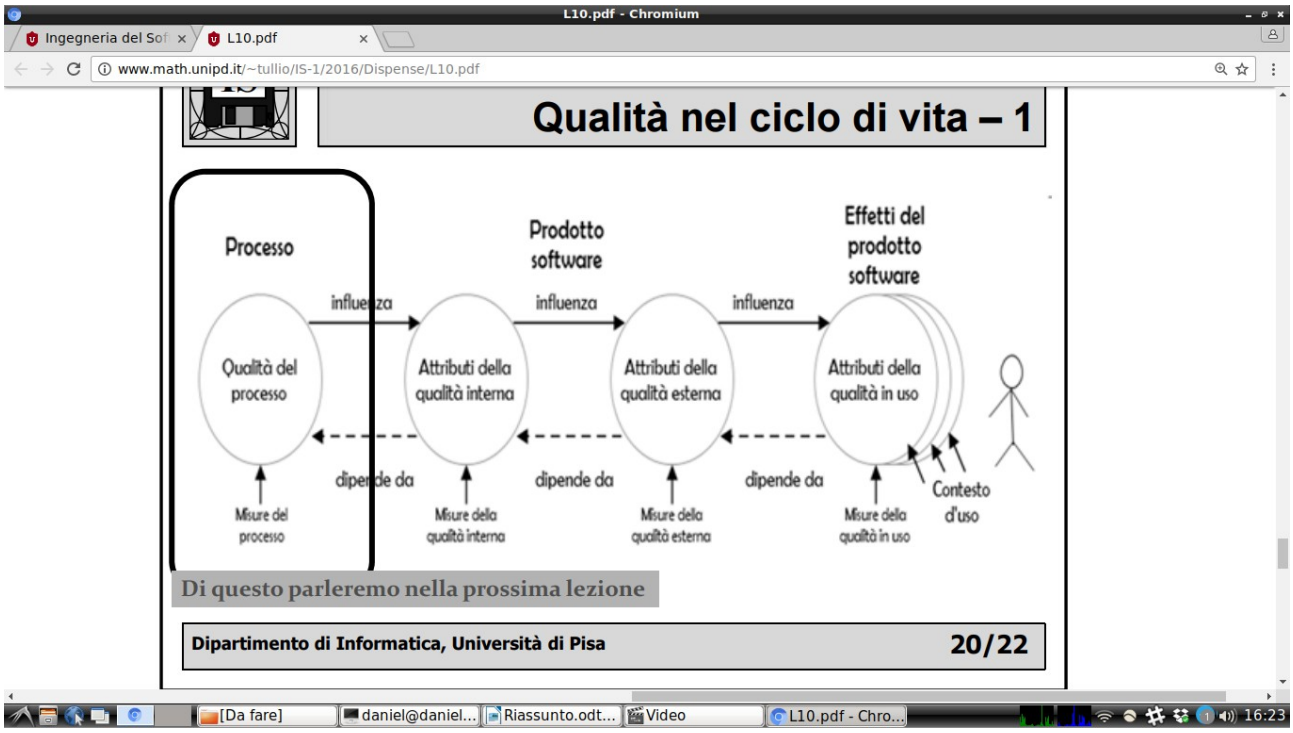
    - **caratteristiche:**
      - **Funzionalità: (Visione esterna)**
        - il prodotto fa quello che ci aspettiamo
        - dimostrabile tramite test
          - attraverso il tracciamento
      - **Affidabilità: (Visione esterna)**
        - se lo uso deve fare il suo mestiere
        - dimostrabile tramite combinazione di test e analisi statica
      - **Efficienza: (Visione esterna)**
        - possa fare il suo mestiere senza chiedere cose smodate
          - es: un programma (Eclipse) mi chiede 4GB di RAM, quando se un PC arriva a 512MB di RAM è già tanto
      - **Usabilità: (Visione esterna)**
        - comprensibile agli utenti
      - **Manutenibilità: (Visione interna)**
      - **Portabilità: (Visione interna)**
        - ho software, scritto una volta
        - nota che per ogni cambio, ho i test di regressione da eseguire!
      - **Qualità in uso:**
        - **relativa alla percezione dell'utente finale**
          - l'utilizzatore **vuole un prodotto che lo aiuti ad essere:**
            - più **produttivo**
            - più **efficacie**
            - soddisfatto usandolo
            - sicuro che non succedano cose spiacevoli mentre lo uso

    - **lo standard ISO/IEC 9126 propone inoltre 3 visioni della qualità:**
      - **visione esterna:**
        - **qualità che vediamo quando l'oggetto software è in esecuzione**
      - **visione interna:**
        - **qualità che vediamo quando l'oggetto software non è in esecuzione**
        - **deriva da scelte di progettazione, codifica, verifica e che si vede solo tramite revisione critica**
      - **qualità in uso:**
        - **relativa alla percezione dell'utente finale**

Fine lezione T12

Lezione T13 – Data: 18/11/2016 – Titolo: Qualità di prodotto_parte2

- **Qualità nel ciclo di vita:**
  - **la qualità del processo sta a monte:**
    - prima dunque rispetto alla qualità del prodotto
    - **influenza gli attributi della qualità interna**
  - **la qualità interna influenza gli attributi della qualità esterna**
  - **la qualità esterna influenza gli attributi della qualità in uso**



- **Metriche e misurazioni:**
  - **Una misurazione quantitativa è l'uso di una metrica** (← glossario) **per assegnare un valore su una scala predefinita.**
  - **Il processo di valutazione, per costruire un sistema che valuti:**
    - **fisso le metriche**
    - **stabilisco come interpretare le misure:**
      - **stabilendo i valori ottimali, accettabili, non accettabili**
    - **effettuo la misurazione:**
      - voglio che le misurazioni siano automatiche
    - **effettuo la valutazione**
      - **è una verifica quantificata**
    - **criteri di accettazione:**
      - **sulla base dei criteri di accettazione scelti**
        - quando accetto ci può essere spazio per un **feedback**
          - feedback:
            - è una valutazione retrospettiva
            - mi aiuta a fare **PDCA**

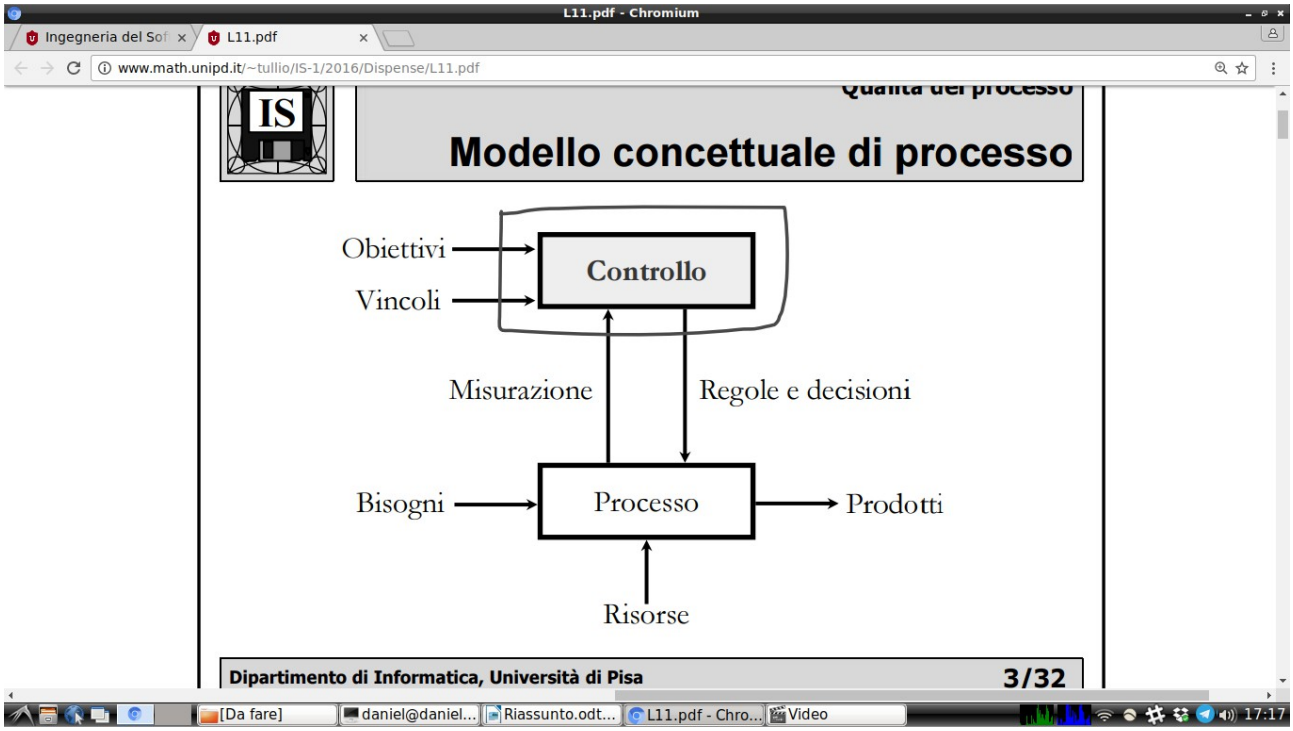
Fine lezione T13

Lezione T14 – Data: 24/11/2016 – Titolo: Qualità di processo

Qualità di processo:

- **Risalire dal prodotto al processo:**
  - **la qualità del processo sta a monte**
    - da tubi sporchi non esce acqua pulita…
  - **per ottenere prodotti di qualità è necessario avere processi di qualità**
  - **è necessario controllare i processi, per assicurarci che i prodotti dei processi siano quelli desiderati:**
    - **per fare ciò sono necessari dei sensori e degli attuatori di una self driving car**
    - **il controllore del processo è la quality assurance**
    - **i sensori sono delle misurazioni effettuate sul processo**
      - **vogliamo misurazioni preventive, il cui obiettivo è il miglioramento continuo (PDCA)**
    - **gli attuatori sono regole e decisioni atte a migliorare il processo**

  - **la qualità in un processo entra attraverso:**
    - **vincoli:**
      - **controllo che tutte le cose fatte dal processo rispettino le norme prefissate**
    - **obiettivi:**
      - **rispetto all'efficacia**
      - **e all'efficienza**



- **ISO 9000 / 9001:**
  - il concetto di qualità di processo fu introdotto dai big spender, per avere delle classifiche oggettive di fornitori
    - il compratore può convincersi che quello che compra sia cosa di qualità
  - nel 1987 nacque ISO 9000
  - **è una famiglia di standard che riguardano i sistemi di gestione della qualità**
    - **stabilisce se ho un way of working di qualità**
      - **parla quindi di processi**
  - **non si occupa solo di software**
  - **ISO 9000 introduce i fondamenti e un glossario**
  - **ISO 9001 si occupa di calare la visione di ISO 9000 nei sistemi produttivi, introducendo dei requisiti ben specifici**
  - **esistono enti che si occupano di certificare il rispetto di tali requisiti**
    - **ISO 9001 è dunque anche una certificazione**

- **Sistema Gestione Qualità (SGQ):**
  - **La gestione della qualità, intesa come funzione aziendale, dovrebbe stare nella parte alta, non nelle foglie, riferendo direttamente alla direzione**
  - **deve garantire la qualità in modo trasversale ai vari settori**
  - **è documentato tramite:**
    - **una politica per la qualità:**
      - cioè le motivazioni che stanno alla base delle scelte sulla qualità
    - **il manuale della qualità:**
      - **definisce il sistema di gestione della qualità di un'organizzazione**
      - **adotta una visione operativa (orizzontale, ad alto livello)**
    - **il piano della qualità:**
      - **è una concretizzazione del manuale della qualità che adotta una visione strategica (verticale)**
      - **definisce gli elementi del sistema di gestione della qualità e le risorse che devono essere applicate in uno specifico caso (prodotto, processo o progetto)**
      - **può avere valenza contrattuale**
      - nel progetto didattico, il piano della qualità è integrato nel piano di qualifica

- **Strumenti di valutazione:**
  - alcuni strumenti per il miglioramento e la valutazione della qualità dei processi:
    - **SPY** (Software Process assessment & improvment)
    - **CMMI**
    - **SPICE**

  - **SPY:**
    - **rappresenta una valutazione oggettiva dei processi di una organizzazione**
    - **fornisce un giudizio di maturità e individua azioni migliorative**

  - **CMMI:**
    - **misura due cose:**
      - **Capability:**
        - **misura di quanto è adeguato un singolo processo, per gli scopi per cui è stato definito**
        - **è una caratteristica di un singolo processo**
        - **è il min di tutti i processi che ho**
          - se ho 10 processi, per valutarne il livello CMM prendo il min
      - **Maturity:**
        - **misura di quanto è governato l'intero sistema dei processi dell'azienda**
        - **è una caratteristica di un insieme di processi**
        - **risulta dall'effetto combinato delle capability dei processi coinvolti**
        - **l'intelligenza dei processi di un'organizzazione si chiama governance**
          - **la governance ha interesse ad alzare il livello di maturity dell'organizzazione**
    - **definisce 5 livelli di maturità:**
      - **Level 1: Initial**
        - **i processi sono imprevedibili, poco controllati e poco reattivi**
      - **Level 2: Managed**
        - **viene applicata la "D" (Do)**
        - **i processi sono gestiti per progetto**
        - **sono spesso reattivi**
          - il verificatore sta alla fine del tubo
        - la differenza tra i livelli 1 e 2:
          - nel livello 2, faccio delle cose perché c'è un piano che dice che devo farle
      - **Level 3: Defined**
        - **viene applicata la "P" (Plan)** (pianificazione di miglioramento)
        - **i processi sono gestiti per organizzazione**
        - **sono proattivi**
        - la differenza tra i livelli 2 e 3:
          - nel livello 3, gestione a livello organizzativo e non solo di progetto
      - **Level 4: Quantitatively Managed**
        - **viene applicata "PC" (Plan, Check)**
        - **i processi sono misurati e controllati**
          - verifica se le azioni migliorative fatte hanno prodotto effetto positivo o no
      - **Level 5: Optimizing**
        - **viene applicata "PDCA) (Plan, Do, Check, Act)**
        - **tutte le cose che faccio sono finalizzate al miglioramento**

- **SPICE:**
  - il resto del mondo (e in particolare l'ISO) ha invece definito ISO/IEC 15504
  - **scompare il concetto di "Maturity"**, presente invece in CMMI
    - **non guarda il bottom**
  - **è più difficile da attuare**
  - **è più "gentile" nell'esito**
    - **vengono utilizzati valori intermedi**, a differenza di CMM:
      - not implemented
      - partial implemented
      - largely implemented
      - fully implemented
  - **la scala prevede 6 livelli:**
    - **sono i 5 di CMM + incomplete**
    - **incomplete**
    - **performed**
    - **managed**
    - **established**
    - **predictable**
    - **optimizing**

Fine lezione T14

Lezione T15 – Data: 15/12/2016 – Titolo: Verifica e Validazione: Introduzione

Verifica e Validazione: Introduzione

- **Verifica:**
  - **E' un processo di supporto**
    - è una “chiamata di funzione” dal processo primario di sviluppo
      - **da in output un valore quantitativo su come siamo messi**
  - **E' un insieme di attività che svolgo durante tutto lo sviluppo:**
    - **più fasi ho, più opportunità ho per fare Verifica**
    - **attività che svolgo ad ogni uscita parziale (=fase) per non trascinare a lungo eventuali non conformità**
    - **è presente ovunque e non ha output tangibili**
      - **la verifica affianca inoltre tutti gli stati del ciclo di vita, quindi non corrisponde ad attività che fanno avanzare i processi**
    - **per assicurarci che stiamo lavorando bene**, nel modo corretto, nel mentre, non dopo!
    - **nell’analisi dei requisiti:**
      - mi assicuro di non essermi dimenticato niente
        - necessità e **sufficienza dei requisiti**
    - **nella progettazione:**
      - mi chiedo se **la soluzione che ho ideato soddisfa i requisiti**
    - **nella codifica:**
      - dev’essere scritto e dimostrato che **il codice soddisfa i requisiti**

  - **E' rivolta ai processi:**
    - **viene svolta sui loro prodotti per accertare il rispetto di regole, convenzioni e procedure**

  - ***Did i build the system right?***

  - **Per fare verifica e validazione ho bisogno di un piano di azione**
    - È qualcosa che interrompe il lavoro
      - non lo ferma

- **Validazione:**
  - **E’ un processo di supporto**
    - è una “chiamata di funzione” dal processo primario di sviluppo
  - **E' un insieme di attività tramite le quali possiamo dire che un prodotto finale è conforme/efficacie**
    - possiamo cioè dire che soddisfa i requisiti

  - **Attuiamo il processo di validazione una volta sola, sul prodotto finale:**
    - **rivolta ai prodotti finali:**
      - accertiamo che un prodotto finale soddisfa i requisiti

  - ***Did i build the right system?***
    - **lo dice il cliente se è quello giusto**

  - **Per fare verifica e validazione ho bisogno di un piano di azione**

  - Come si arriva a fare validazione in maniera sensata?
    - **dev’essere una “self fulfilling prophecy”**
      - **è una cosa che dico prima ma so già che sarà vera**
      - non dev'essere quindi un’attività a rischio
    - **svolgendo attività preventive che mi dicano che non ho sbagliato a fare le cose**
      - prende il nome di **Verifica**

  - la validazione **non la facciamo sui requisiti piccoli**, esplosi
    - **la facciamo sui gruppi grandi**

- **Qualifica = V & V**
  - **Verifica + Validazione = Qualifica**
  - è il raggiungimento di informazione quantitativa sul soddisfacimento dei requisiti
  - la **qualifica**, data dalla verifica e validazione (← glossario) **ci indica se stiamo soddisfacendo i requisiti**

- **Forme di verifica:**
  - **Si può fare in due forme:**
    - **Analisi statica:**
      - **viene fatta prima dell'analisi dinamica**
      - **non richiede l'esecuzione del prodotto software in alcuna sua parte**
      - **si applica ad ogni prodotto intermedio per tutti i processi attivi nel progetto**
        - **non solo software**
      - **due tecniche per fare analisi statica:**
        - **metodo di lettura (desk check):**
          - **svolto da un umano che controlla se l'oggetto della verifica rispetta determinate caratteristiche, valutandone consistenza, completezza, correttezza**

          - **due metodi di lettura:**

            - **inspection:**
              - **eseguo una lettura mirata del prodotto in esame**
              - **focalizzo la ricerca sugli errori noti, più probabili**
              - **può generare falsi negativi**
              - **è molto più rapido del walkthrough**
              - **si suddivide nelle seguenti attività:**
                - **pianificazione**
                - **definizione di una checklist**, cioè di una lista di controllo
                  - **evolve nel tempo**
                - **lettura**
                - **correzione dei difetti**

            - **walkthrough:**
              - **eseguo una lettura critica del prodotto in esame**
              - **è una lettura a largo spettro, senza l'assunzione di presupposti**
              - **può generare falsi positivi**
              - **tipologia molto costosa e che rischia di essere soggettiva**
              - **si suddivide nelle seguenti attività:**
                - **pianificazione**
                - **lettura**
                - **discussione**
                - **correzione dei difetti**

        - **metodo formale:**
          - **svolto da automi**

    - **Analisi dinamica:**
      - **il software, o una sua parte, viene eseguito**
      - **se alla base c'è una solida analisi statica, l'analisi dinamica dovrebbe essere conforme alle attese**

Fine lezione T15

Lezione T16 – Data: 16/12/2016 – Titolo: Verifica e Validazione: Analisi statica

Verifica e Validazione: Analisi statica

- **Premessa:**
  - **un software di buona qualità deve possedere:**
    - **tutte le capacità funzionali specificate nei requisiti**
    - **tutte le caratteristiche non funzionali necessarie al buon funzionamento del sistema**
    - **queste caratteristiche devono essere verificabili in modo certo**
  - **occorre scegliere un linguaggio di programmazione bilanciando:**
    - **le funzionalità del linguaggio**
      - cioè il potere espressivo (es: Thread in Java e in C)
    - **integrità del linguaggio**
      - per i costi di verifica
  - **è bene adottare uno standard di codifica che tenga conto della verifica**
    - **accompagnando la verifica durante la produzione di codice, e non in modo retrospettivo**
    - **il costo di rilevazione e correzione di un errore è tanto maggiore quanto più avanzato è lo stadio di sviluppo**

    - **per assicurare comportamento predicibile:**
      - senza side effects
        - diverse invocazioni della stessa funzione producono risultati diversi
        - ad esempio con l'utilizzo di allias
      - ordine di elaborazione e inizializzazione
        - ad esempio, l'attivazione dei thread in Java è fonte di impredicibilità
      - modalità di passaggio dei parametri
        - per valore, per riferimento
    - **per usare soliti e collaudati criteri di programmazione:**
      - riflettere l'architettura nel codice
        - programmazione strutturata per esprimere componenti, moduli, unità come da progettazione, e facilitare il riuso
      - separare le interfacce dall'implementazione
        - stabilendone i contratti
      - massimizzare l'incapsulazione
        - information hiding
        - usare membri privati e metodi pubblici per l'accesso
    - **per ragioni pragmatiche:**
      - l'efficacia dei metodi di analisi è funzione della qualità di struttura del codice
        - esempio:
          - un modulo con un solo punto di ingresso e un solo punto di uscita è più facilmente analizzabile per il suo effetto sullo stato

Fine lezione T16

Lezione T17 – Data: 19/01/2017 – Titolo: Verifica e Validazione: Analisi dinamica

Verifica e Validazione: Analisi dinamica

- **Analisi dinamica = test (prova)**
  - **il software, o una sua parte, viene eseguito**
  - **se alla base c'è una solida analisi statica, l'analisi dinamica dovrebbe essere conforme alle attese**
  - **è importante notare che il testing viene fatto sempre**
    - **se non lo fa il fornitore, lo farà l'utente**
  - **produce una misura della qualità del sistema**
  - **la pianificazione dei test deve avvenire a monte della codifica, durante la progettazione del sistema**
  - **i test hanno la caratteristica della non esaustività**
    - **posso testare solamente un insieme finito di casi**
  - **oggetto del test può essere:**
    - **il sistema nel suo complesso**
    - **parti di esso**
      - in relazione funzionale, d'uso, di comportamento, di struttura
      - per i test di integrazione
    - **singole unità**
      - per i test di unità
  - **il Piano di Qualifica risponde alla domanda: quali e quanti test?**
  - **i testi sono costosi**
    - richiedono molte risorse (tempo/persone/infrastrutture)
    - necessitano di un processo definito
    - richiedono attività di ricerca, analisi e correzione
  - **un test può rilevare la presenza di malfunzionamenti, ma non può dimostrarne l'assenza**
    - **non esiste un modo automatico per dire che un programma è corretto.**

- **Terminologia:**
  - **Failure:**
    - **quando un’esecuzione è difforme dalle attese**
      - esempio: **un output sbagliato**
    - **non è qui che è ragionevole intervenire**
    - **si manifesta una failure se si ricopre un determinato stato erroneo**

  - **Error:**
    - **è uno stato del sistema, che se attivato produce una failure (malfunzionamento)**
      - ci sono nei cammini o delle istruzioni sbagliate
        - se non ci vado sopra, non vedo l’errore

  - **Fault:**
    - **c’è uno stato erroneo perché c’è un fault**
      - è dunque causa di un errore
    - **può riguardare:**
      - **il modo le cose funzionano:**
        - si è rotto il PC, la ram…
      - **c’è un guasto logico**
        - **abbiamo progettato male/programmato male**

- **Organizziamo il testing nelle seguenti classi:**
  - **test case:**
    - **è una tripla <input, output, ambiente>**
  - **test suite:**
    - **è un insieme di test**
  - **test:**
    - **una procedura di test**

- **Fattori da bilanciare:**
  - **vale la regola del rendimento decrescente:**
    - **man mano che aumento lo sforzo, il rendimento cresce inizialmente ma raggiunta una certa soglia non cresce più**
  - la strategia di test richiede di bilanciare:
    - **la quantità minima di casi di test sufficienti a fornire certezza adeguata sulla qualità del prodotto**
      - fattore governato da **criteri tecnici**
      - **Come ridurre il numero dei test:**
        - **ricorrendo alla tecnica delle classi di equivalenza**
        - **ci interessano:**
          - **Illegali**
          - **Legali interni**
          - **Confine inferiore**
          - **Confine superiore**
    - **la quantità massima di sforzo, tempo e risorse disponibili per la verifica**
      - fattore governato da **criteri gestionali**

- **Test e ambiente di prova:**
  - **La ripetibilità di un test è requisito essenziale; per ciò bisogna sempre:**
    - **tener conto dell'ambiente di esecuzione**
    - **specificare le PRE e POST condizioni**
    - **descrivere le procedure per eseguire il test e per analizzarne i risultati**

  - **Servono 3 strumenti per fare test:**
    - **Driver:**
      - **ha controllo sull'esecuzione dei test**
      - **i test vengono "chiamati" dai driver**
    - **Stub:**
      - **componente passiva fittizia per simulare una parte del sistema**
      - **non è oggetto di test**
      - viene comandato dal driver
    - **Logger:**
      - **scrive l'esito della prova su un file**
      - **componente non intrusivo**

- **Tipologie di test:**
  - **Collaudo o Test di accettazione:**
    - **stabiliti durante l'analisi del capitolato d'appalto**
    - **è un'attività esterna, supervisionata dal committente, nella quale si dimostra la conformità del prodotto (si accerta il soddisfacimento dei requisiti utente)**
    - **al collaudo segue il rilascio del prodotto e la fine della commessa**

  - **Test di sistema:**
    - **stabiliti durante l'analisi dei requisiti**
    - **per accertare la copertura dei requisiti software**
    - **sono inerentemente funzionali**
      - non hanno bisogno di conoscere la logica del sistema

  - **Test di integrazione:**
    - **stabiliti durante la progettazione logica**
    - **stanno a livello di componente e integrano il funzionamento di più unità**
    - **servono per verificare il sistema in modo incrementale**
      - **è bene assemblare le parti in modo incrementale**
        - **aggiungendo una parte nuova ad un insieme ben verificato, i difetti rilevati in un test d'integrazione saranno probabilmente dovuti alla parte nuova, facilitando così la ricerca di quale parte sia da correggere.**
    - **possiamo individuare due principali strategie di integrazione:**
      - **bottom-up:**
        - **si sviluppano e integrano prima le parti con minore dipendenza funzionale (fan-out) e maggiore utilità (fan-in)**
        - **risparmio stub**
        - **le funzionalità di alto livello compaiono più tardi**
      - **top-down:**
        - **si sviluppano prima le parti più esterne**
        - **le funzionalità ad alto livello vengono sviluppate sin da subito**
        - **si può mostrare al committente una bozza del sistema**

- **Test di unità:**
  - **stabiliti durante la progettazione di dettaglio**
  - **è un'attività di analisi dinamica che verifica le singole unità software**
  - **vengono svolti con un alto grado di parallelismo**
  - **può essere una responsabilità del programmatore, del verificatore o meglio ancora di un automa**
  - **sono i più numerosi**

  - **Ci sono due categorie di test di unità:**

    - **test funzionali (black-box):**
      - **da eseguire prima dei test strutturali**
        - **vanno integrati con i test strutturali**
      - **test a scatola chiusa**
      - **utilizzano input capaci di provocare output attesi**
      - fanno riferimento alla specifica dell'unità

    - **test strutturali (white-box):**
      - **da eseguire dopo i test funzionali**
      - **test a scatola aperta**
        - **verificano la logica interna del codice**
      - **persegue la massima copertura del codice**
        - **un concetto fondamentale nei test di unità è dato dalla copertura (coverage):**
          - **si intende la percentuale di codice che un test è in grado di eseguire**
          - **ciascun test deve essere progettato per attivare ogni cammino di esecuzione all'interno del modulo**
            - **al crescere del numero di condizioni all'interno di una decisione il numero di test necessario a massimizzare il condition coverage diventa proibitivo**

          - **alcuni criteri di copertura:**

            - **Function coverage:**
              - **quante funzioni (sotto-procedure) sono state eseguite?**

            - **Statement Coverage:**
              - **quanti statement sono stati eseguiti?**
              - **è cumulativo:**
                - **conto se ho eseguito almeno una volta tutti gli statement, considerando tutti i cammini possibili**
              - **è meno potente del branch coverage**
                - esempio: istruzione posta nel ramo condizionale sbagliato

            - **Branch Coverage:**
              - **quanti rami del programma sono stati eseguiti?**
              - **quando ciascun ramo del flusso di controllo viene attraversato, complessivamente, almeno una volta**

- **Test di regressione:**
  - **non vanno pianificati, la regressione dev'essere studiata ad hoc**
  - **rappresentano l'insieme dei test necessari ad accertare che la correzione introdotta, fatta a seguito di un test fallito, non abbia introdotto errori nelle parti del sistema che dipendono da essa**
    - **il rischio aumenta con alto accoppiamento e bassa incapsulazione**

Fine lezione T17

Lezione T18 – Data: 27/02/2017 – Titolo: Misurazioni software

Metodi e obiettivi di quantificazione:

- **Misurare:**
  - **definizione:**
    - **la misurazione è il processo che assegna numeri o simboli a entità del mondo reale. Risultato di una misurazione è una misura.**
  - **vogliamo misurare quanto siamo vicini agli obiettivi**
    - **vogliamo misurare in modo proattivo, e non reattivo**
    - questi ultimi devono dunque essere quantificabili
  - **le misurazioni devono essere oggettive, garantendo cioè:**
    - **ripetibilità**
    - **confrontabilità**
    - **confidenza**
  - **standard ISO/IEC 15939 per la misurazione dei bisogni informativi**

- **Cosa ci interessa misurare:**
  - **processi**
    - **per valutarne la qualità**
  - **progetti**
    - **per effettuare stime, preventivi e consuntivi**
  - **prodotti**
    - **per valutarne la qualità**
  - **risorse**
    - **per capirne il consumo**

- **Categorie di attributi:**
  - **gli attributi a cui le misurazioni assegnano valori, possono essere:**
    - **interni:**
      - riguardano la qualità interna, misurabili rispetto all'entità
    - **esterni**
      - riguardano la qualità esterna, misurabili rispetto all'ambiente

Fine lezione T18

Fine.

Glossario:

- **Efficienza ed Efficacia:**
  - **Efficienza:**
    - = Ottimizzazione
      - di energia
      - di tempo/persona
      - di materiale
    - La sua funzione obiettivo è min
      - vogliamo minimizzare la quantità di risorse
    - Inversamente proporzionale alla quantità di risorse impiegate nell'esecuzione delle attività richieste
    - E' un indicatore quantitativo di quanto sto consumando

  - **Efficacia:**
    - = Conformità
      - rispetto a regole
      - rispetto alle attese
      - rispetto ai bisogni
    - La sua funzione obiettivo è max
    - la strategia con la quale il fornitore garantisce la conformità viene fissato nel Piano di Qualifica
      - dove **Qualifica = Verifica & Validazione**

  - i due termini confliggono: l'efficacia sottintende il verbo "fare", mentre la massima efficienza è proprio il non fare nulla! Bisogna dunque badare a trovare un compromesso, possibilmente in modo sistematico. Ma sappiamo che esiste un ciclo virtuoso tra sistematizzazione ed esperienza. Ecco allora che si ricorre alla **best practice** (← glossario)

- **Best practice:**
  - prassi (modo di fare) che per esperienza e per studio abbia mostrato di garantire i migliori risultati in circostanze note e specifiche.
  - il miglior modo di fare ciò che devo fare

- **Stakeholder:**
  - portatori di interesse
  - l’insieme di persone a vario titolo coinvolte nelle decisioni importanti riguardanti il ciclo di vita del SW, che hanno influenza sul prodotto o sul processo.

- **Fase:**
  - segmento temporale che cattura stati o transizioni di un **ciclo di vita** nel quale si svolgono specifiche (e quindi univoche) attività
    - **mal si adatta** con l’adozione del modello di sviluppo **incrementale**, che invece precede la frequente ripetizione di alcune attività.
  - identifica un intervallo di tempo definito, una volta passato non c’è più
    - esempio:
      - le fasi lunari
        - prima crescente…
        - ...poi calante

- **Periodo:**
  - Definisce delle suddivisioni di tempo, con cui è stato suddiviso il progetto, che possono anche essere concorrenti o ripetute

- **Prototipo:**
  - serve per provare a scegliere soluzioni
  - **può essere “usa e getta” (nel caso delle iterazioni) oppure fornire stati di incremento** (**baseline**)
  - **doppia funzione:**
    - una positiva:
      - **ha un valore:**
        - associato ad un senso di incremento
          - è un pezzo di soluzione
            - una **baseline** (← glossario)
        - **può aiutarmi a capire meglio il problema**
    - una neutra/negativa:
      - **ha un rischio:**
        - **non concorre a far parte del prodotto finale**
        - **mi fa pensare ad un’iterazione**

- **Baseline:**
  - “punto d’appoggio”
  - è un punto di arrivo tecnico dal quale non ho bisogno di retrocedere
  - **è il prodotto tangibile, associato alle milestone** (← glossario)
    - può essere un documento, un manuale, uno strumento di verifica, o tutte e 3, ecc...
  - **rappresenta un risultato sul quale baserò il successivo incremento**
    - è un’approssimazione dell’esito finale
  - contiene una parte in cui ne vengono descritte le caratteristiche
    - quali **CI** (← glossario) sono cambiati
  - l’esistenza di baseline ben identificate **permette:**
    - **riproducibilità**
    - **tracciabilità**
    - **analisi e confronto**
  - **sta in una repository** (← glossario)**:**
    - **deve avere configurazione** (← glossario)
    - **deve avere una versione** (← glossario)
  - **usa e getta != baseline**
    - sono due concetti opposti
    - una baseline è per starci

- **Milestone:**
  - **“pietra miliare”**
  - **indica importanti traguardi intermedi nello svolgimento del progetto.**
  - E’ un punto nel calendario
    - **è un punto nel tempo associato ad un valore strategico**
      - o perché c’è un **obbligo**…
        - es: il cliente vuole vedere come le cose avanzano
      - ...o perché c’è **un’opportunità**
  - **Un progetto nasce (anche) in funzione di una milestone: quella del cliente**
    - ultima data di attesa per consegnarlo
      - “voglio il progetto X entro il 4 Settembre”
  - è uno **strumento a calendario che misura l’avanzamento di progresso**
    - ma non deve essere usato per accorgerci che siamo in ritardo!
  - **Associo alle milestone un avanzamento tecnico, che se non sono usa e getta posso chiamare baseline**
  - Buone qualità di milestone:
    - **specifiche** per obiettivi
    - **delimitate** per ampiezza e ambizioni
    - **incrementali** per contenuti
    - **coerenti e rilevanti** per la strategia di progetto
    - **misurabili** per quantità di impegno necessario
    - **traducibili in compiti assegnabili**
    - **raggiungibili**
    - **puntuali** rispetto alle esigenze di calendario
    - **dimostrabili** agli stakeholders

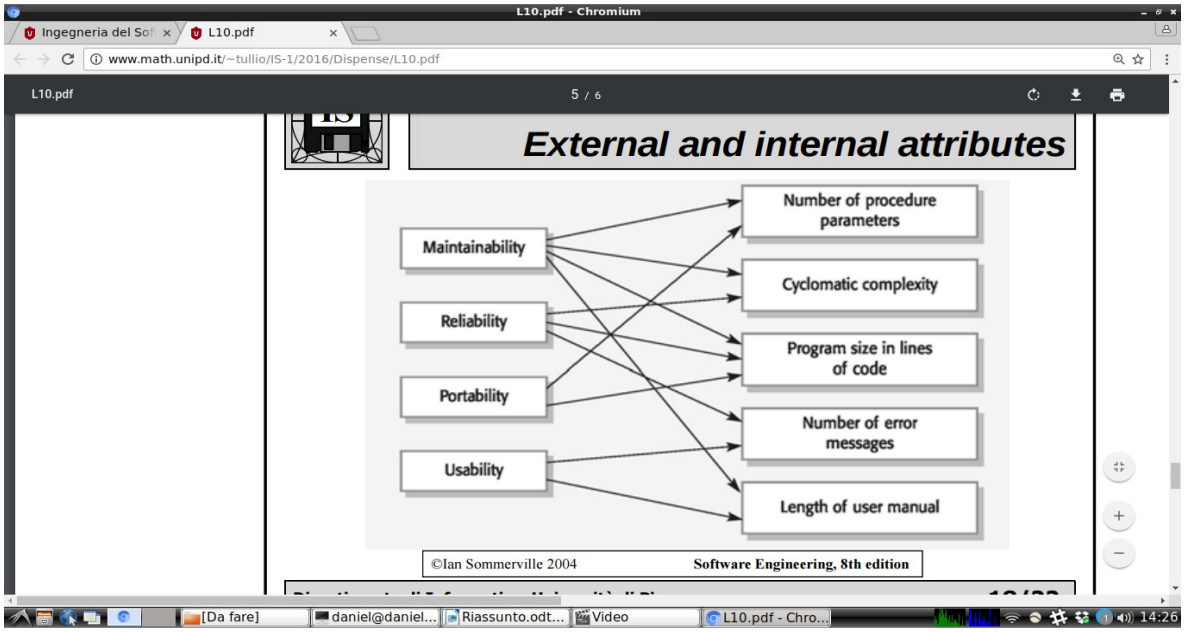
- **Issue Traking:**
  - Derivato dall'ambito del *Service Management*
  - ITS, issue traking system, si stratta di un sistema informatico che gestisce e registra delle liste di richieste di assistenza o di problemi, organizzato secondo le necessità di chi offre il servizio.

- **Tracciamento:**
  - **Viene definito "tracciamento" l'incontro dei bisogni e dei requisiti**
    - non farò mai una cosa (soluzione) se non soddisfa i requisiti
  - **E' parte fondamentale dell'analisi statica**
  - **può essere altamente automatizzato**
  - **È uno strumento importantissimo di misurazione**
    - **fornisce un dato quantitativo**
    - indica quanti requisiti sono stati soddisfatti
      - puntiamo al 100% dei requisiti
    - **più requisiti soddisfa, più il prodotto è efficacie/conforme**
      - **ogni requisito non soddisfatto andrà giustificato!**
  - **E' una verifica atta a dimostrare due caratteristiche di una soluzione:**
    - **completezza:**
      - **tutti i requisiti sono soddisfatti**
    - **economicità:**
      - **nessuna funzionalità superflua**
      - **nessun componente ingiustificato**
  - **Ha luogo su ogni passaggio dello sviluppo (ramo discendente del modello a "V") e su ogni passaggio della verifica (ramo ascendente).**
    - **su ogni passaggio dello sviluppo (ramo discendente):**
      - **mi assicuro di non dimenticarmi niente, man mano che scendo**
      - quando faccio l'analisi dei requisiti traccio se ho preso tutto e se ciò che ho è ciò che serve
      - quando progetto controllo se i requisiti li ho presi tutti in carico, controllando di non fare cose inutili
      - quando progetto nel dettaglio, mi chiedo a quali requisiti appartengono le foglie del dettaglio
    - **su ogni passaggio della verifica (ramo ascendente)**
      - **mi dice che ho fatto ciò che dovevo fare**
      - nei test di unità, mi chiedo hai testato tutte le unità?
  - **Tracciare un requisito significa motivarne l'esistenza**
    - **spiegando qual è l'origine di tale requisito**
    - **badando a garantire la necessità e la sufficienza di ogni requisito.**
  - **Si può usare una matrice, un grafo o una qualsiasi struttura dati appropriata.**
  - **I principali tracciamenti all'interno di un prodotto sono:**
    - **requisiti utente (capitolato) ↔ requisiti software (AR)**
    - **requisiti software (AR) ↔ descrizione di componenti (ST)**
    - **test di unità ↔ moduli della progettazione di dettaglio (DP)**
    - **test di integrazione ↔ componenti architetturali (ST)**
    - **test di sistema ↔ requisiti software (AR)**
    - **test di accettazione ↔ requisiti utente (capitolato)**

- **Design pattern:**
  - **Soluzione progettuale generale a un problema ricorrente.**
  - **definisce una funzionalità lasciando gradi di libertà d'uso**
    - ha corrispondenza precisa nel codice sorgente
  - concetto promosso da C. Alexander (un vero architetto)
    - **l'architettura fa emergere degli stili**
    - **gli stili si veicolano attraverso dei pattern**
    - **per la progettazione a livello sistema si usano pattern architetturali**(← glossario)

- **Framework:**
  - **= lavorare dentro ad un quadro**
  - **molto importante per la progettazione**
    - **danno forma ad un' architettura** alla quale io possa aderire
  - **è una struttura di supporto su cui un software può essere organizzato e progettato**
    - **è un insieme di librerie/utilità che aiutano in qualche compito**
    - **è un insieme integrato di componenti software prefabbricate:**
      - **nel mondo pre-OO erano chiamate librerie:**
        - i linguaggi di programmazione del passato non erano strutturati
          - es: C
      - **quando hanno iniziato ad esistere classi e interfacce, si è visto che queste formavano quasi un'architettura**
  - sono **bottom-up perché fatti di codice già sviluppato**
  - sono **top-down se impongono uno stile architetturale**
  - esempi:
    - QT
    - molti importanti esempi nel mondo J2EE e JS
      - Spring per architettura di business con MVC
      - Swing per GUI, ecc

- **Metrica:**
  - **definizione secondo IEEE:**
    - **misura quantitativa del grado di possesso di uno specifico attributo da parte di un sistema, componente, processo**
  - **sistema per dare un significato standardizzato a valori misurati**
    - un valore da solo non significa niente
      - ha significato dentro ad una metrica
  - **è divisa in:**
    - **misurazione**
    - **attribuzione di un significato alle misurazioni**
  - **permettono di quantificare:**
    - **prodotti**
    - **processi**
      - al fine di monitorarli e migliorarli
  - **la selezione delle metriche (e la scelta dei criteri di accettazione) parte dall'analisi dei requisiti di qualità e dei vincoli di costo**
  - **l'uso di una metrica assume che:**
    - **si può misurare un certo attributo del software**
    - **esiste una relazione tra ciò che vogliamo misurare e ciò che vogliamo sapere**
      - ad esempio, generalmente è possibile misurare solo attributi interni del software, mentre molto spesso ciò che ci interessa sono gli attributi esterni
        - esempio: voglio conoscere il grado di manutenibilità del software (attributo esterno); mi serve conoscere alcuni attributi interni, come ad esempio il numero di righe di codice, il numero di parametri per la procedura, l'ampiezza del manuale utente, … (vedi figura)
    - **la suddetta relazione è stata formalizzata e validata**
  - **nella repository** (← glossario) **voglio far entrare solamente prodotti che soddisfino le metriche**
  - esempi:
    - parlando di project management, le risorse le misuriamo attraverso la metrica “ore/persona”
    - sono misurabili anche i documenti
      - ad esempio, l’indice Gulpease per la leggibilità del documento



- **Analisi:**
  - attività con la quale facciamo verifica

- **Test:**
  - prova che svolge un pezzo di analisi dinamica

- **Componente:**
  - **integra più unità**

- **Unità:**
  - **è la più piccola quantità di software che conviene verificare, testare da sola**
  - **deducibile dalle foglie della progettazione di dettaglio**
  - **non linee di codice, ma entità di strutturazione (procedura, classe, package)**
  - **ha lo scopo di soddisfare un certo numero di requisiti**
    - **non deve avere troppi requisiti**

- **Modulo:**
  - **è parte dell'unità**
  - **componente elementare di progetto di dettaglio**
    - **non necessita quindi di test di unità**

- **Preventivo:**
  - **è un calcolo fatto su un piano che deve ancora svilupparsi**
    - nel Piano di Progetto il fornitore prevede di impegnare X ore ad un costo di Y €
  - è il preventivo dell'impegno da quando abbiamo avuto l'esito (positivo) dell'RR
    - le ore prima vengono definite **ore di investimento** non rendicontato

- **Consuntivo:**
  - **rappresenta quanto ho effettivamente speso**
  - **divide il tempo in due parti:**
    - **consuntivo di periodo:**
      - **fino a dove sono, calcolo il consuntivo di periodo**
    - **preventivo a finire:**
      - **sul futuro, aggiorno il preventivo e lo chiamo preventivo a finire**

- **Preventivo a finire:**
  - **è l'aggiornamento del preventivo iniziale, detratto ciò che è passato**

- **Consuntivo di periodo:**
  - **rappresenta il tempo rendicontato che consumo dal preventivo**

- **Riuso:**
  - **assembliamo cose già esistenti, buone, secondo una logica sana, secondo una best practice (← glossario)**
    - **ottenendo minor costo realizzativo**
    - **ottenendo minor costo di verifica**

  - **il riuso ha due forme:**
    - **sistematico:**
      - **è positivo**
      - **non significa che sia identica, ma significa che sia adattabile al nuovo compito**
      - **maggior costo**
      - **maggior impatto**
    - **occasionale** (opportunistico)**:**
      - **è malvagio**
      - **non mi fa imparare, non so se mi farà realmente avanzare**
      - **copia-incolla di software**
      - **basso costo**
      - **scarso impatto**

  - **problemi:**
    - **<u>progettare per</u> riuso è più difficile**
      - **bisogna anticipare bisogni futuri**
    - **<u>progettare con</u> riuso non è immediato**
      - **bisogna minimizzare le modifiche alle componenti riusati per non perderne il valore**

  - **componenti riusabili possono includere:**
    - **codice sorgente o eseguibile**
    - **specifiche di interfaccia (ad esempio API)**
    - **modelli architetturali (design pattern)**

- **Configurazione:**
  - **Un insieme è composto di parti: quali sono queste parti il modo in cui devono stare insieme è detto "configurazione"**
    - un esempio: il makefile
      - dice che cosa serve per costruire il prodotto finale (l'eseguibile), quali sono i pezzi e dove si trovano e dice che servono solo le cose più recenti

  - **3 significati:**

    - **forma 1: preferenze**
      - **fare in modo che un certo ambiente/strumento abbia un comportamento standardizzato per certi utenti**
        - **per uniformare il modo di lavorare**

    - **forma 2: avere una posizione nota per ogni tipo di elemento di un prodotto**
      - "la cassettiera", **il filesystem del progetto, deve essere ordinato**
        - **documenti**
        - **manuali**
        - **sorgenti**
        - **test**

    - **forma 3: la configurazione è la costruzione di un prodotto finale**
      - **regole per costruire un prodotto finale a partire da ingredienti che hanno un luogo di abitazione** (locale, nel mio repository, …)
        - **ad esempio un eseguibile**

- **Controllo di configurazione:**
  - **è ciò che serve per gestire la configurazione**
  - **ogni sistema fatto di parti va gestito con controllo di configurazione, secondo regole rigorose**
  - **Le regole di configurazione vanno pianificate**

- **Gestione di configurazione:**
  - **E’ un processo di supporto**
  - **Dev'essere automatizzata**
    - tramite ad esempio strumenti di build

  - **Obiettivi:**
    - **mettere in sicurezza le baseline** (← glossario)
    - **consentire ritorno a configurazioni precedenti**

  - **Attività:**
    - **Identificazione di configurazione:**
      - **quali parti (configuration item, CI (← glossario)) compongono il prodotto**

    - **Controllo di baseline** (← glossario)**:**
      - l’insieme di **CI** (← glossario) consolidato a una specifica **milestone** (← glossario)

    - **Gestione delle modifiche**
      - **tenendo sempre traccia dello stato precedente**

    - **Controllo di versione**
      - (← glossario)

- **Controllo di versione:**
  - **Tutto ciò che è oggetto di manutenzione ha una storia, che va gestita con un controllo di versione**
    - **potrebbe essere necessario tornare ad una versione precedente**
  - **Non controllo la versione del prodotto**
    - **controllo la versione di ogni configuration iteam (← glossario) del prodotto**
  - **Si appoggia su un repository (← glossario)**
  - **Permette a ciascuno di lavorare su vecchi e nuovi CI senza rischio di sovrascritture accidentali**
    - **check-out**
  - **Permette di condividere il lavorato nello spazio comune**
    - **check-in**

- **Repository:**
  - **FS con una root**
  - **DB centralizzato nel quale risiedono – individualmente – tutti i CI (← glossario) di ogni baseline (← parole chiave) nella loro storia completa**
  - **voglio far entrare solamente prodotti che soddisfino le metriche (← glossario) stabilite**

- **Versione, Variante, Release:**
  - **Tutte vanno identificate, pianificate e gestite**
    - identificazione per numero, caratteristiche, modifiche
  - **Versione:**
    - **istanza di prodotto funzionalmente distinta dalle altre**
  - **Variante:**
    - **istanza di prodotto funzionalmente identica ad altre ma diversa per caratteristiche non funzionali**
  - **Rilascio (release):**
    - **istanza di prodotto resa disponibile a utenti esterni**

- **Configuration Item (CI):**
  - **rappresenta un singolo elemento, oggetto delle attività di gestione della configurazione del prodotto**
  - **ogni CI ha una identità unica**
    - ID, nome, data, autore, registro delle modifiche, stato corrente

- **Versionamento:**
  - **@versione**
  - **@controllo di versione**
  - **è un'attività della gestione della configurazione (← glossario)**
  - **le due parole più importanti quando parliamo di versionamento sono:**

    - **repository** (← glossario)**:**
      - **ho bisogno di un file system**
      - **ho bisogno di un sistema per capire cos'è vecchio e cos'è nuovo**

    - **commit:**
      - **in questo modo una persona può affermare di aver completato il proprio compito (task)**
      - **è un atto tracciato:**
      - **è collegata ad un meccanismo in cui candido il mio lavoro, per evitare che nella repository entri spazzatura**
        - verrà accettata attraverso un meccanismo di verifica automatizzata