03/10/2022: Introduzione: metodo e concetti chiave di sviluppo del software

Cose utili:

* Tullio usa solo il Moodle e mette il materiale sul Moodle o sul calendario stesso (tende ad usare siti esterni, comunque accessibili dal Moodle stesso). Non mette tutte le slide (una lezione sì ed un’altra no)
* Legenda del calendario:
  + T 🡪 Teoria
  + P 🡪 Pratica
  + PD 🡪 Progetto Didattico
  + E 🡪 Esercitazione
* Glossario (termini specifici del dominio di interesse) delle parole chiave:
  + Protocollo: in informatica, un insieme di regole o procedure per la trasmissione di dati tra dispositivi elettronici, come i computer. Affinché i computer possano scambiarsi informazioni, deve esistere un accordo preesistente su come le informazioni saranno strutturate e su come ciascuna parte le invierà e le riceverà.
  + Progetto (software) 🡪 Visto (esaminato) da chi lo crea e da chi lo usa. Idee chiave:
    - Sviluppo continuativo
    - Professionale
    - Valutazione qualitativa
    - Collaborativo
    - Monitoraggio

Esso deve essere un mezzo e non un “obiettivo” (goal/aim).

Quest’ultimo, infatti, va compreso; infatti, è visibile a pochi da subito ma l’obiettivo deve essere chiaro a tutti. Non si intende programmazione, quanto piuttosto la *realizzazione di obiettivi* per mezzo dello stesso progetto.

* + Sviluppo: Percorso di realizzazione tra obiettivi *detti* ed obiettivi *raggiunti* per il tramite del software. Questi ultimi rispondono principalmente a dei *bisogni (needs)*, essendo essi stessi il *mezzo* per realizzarli. Si intende tutto ciò attraverso il percorso di progettazione (*design*), di cui il progetto è un prodotto. L’insieme delle regole che lo realizzano è il *paradigma*, seguito dagli informatici che realizzano software e applicano queste regole.
    - Es. Zoom 🡪 Architettura client (inoltrano richieste) – server (soddisfano le richieste presenti) non bastevole; infatti, tutte le richieste sono soddisfatte da un server centrale (senza che l’utente sappia nulla di tutto ciò). Esso è uno strumento

web-based (quindi, una risposta ad un bisogno) 🡪 risultando *portatile*.

Essa è una *piattaforma* (un aggregato che, raduna tanti main(), idealmente compiendo tante cose insieme).

Per approfondire:

<https://www.lavivienpost.com/how-zoom-works/>

Si deriva dalla concezione del Web 2.0 (utenti che creano contenuti); un’architettura tradizionalmente intesa come client-server ha dovuto evolvere per rispondere a nuovi bisogni.

* + - Per realizzare in maniera coesa (interconnessa, espressa come unica entità) le azioni di tanti programmi distinti.
    - Il ruolo dell’informatico è distinguere i bisogni tecnici e i bisogni utente creando un prodotto di *qualità* (quindi seguendo una chiara idea di realizzazione (protocollo), comodo da usare, accessibile, ecc.), risultando dunque professionale. Si cerca di raggiungere lo stato dell’arte (inteso come la qualità in un senso personale) attraverso un costante *autoapprendimento*. Colui che ha gli strumenti e crea è il *practitioner*, continuando ad imparare in autonomia.

04/10/2022: Introduzione (slide omonima): obiettivi, metodo, concetti chiave

(Eventuali approfondimenti listati dal prof:

<http://worrydream.com/refs/Brooks-NoSilverBullet.pdf>

<https://www.acm.org/binaries/content/assets/membership/images2/fac-stu-poster-code.pdf>)

Apprendiamo metodi e pratiche di lavoro alla base della professione informatica.

* Gestire il tempo 🡪 Risorsa scarsa, in quanto soggetto a termini/deadlines
  + Disponibilità, scadenze, conflitti, priorità
* Collaborare 🡪 I compiti da svolgere richiedono molto tempo per un singolo individuo e quindi devono essere suddivisi
  + Fissare obiettivi, dividersi compiti, verificare progressi, riportare difficoltà
* Assumersi responsabilità 🡪 Incarico per garantire una partecipazione equa, di qualità e prendere consapevolezza dei propri doveri
  + Fare quanto pattuito, agire al meglio delle proprie capacità, auto-valutarsi prima di valutare
  + Usare strumenti allo stato dell’arte (quindi, strumenti consolidati e validi nell’ambito di lavoro e di utilizzo)
* Auto-apprendere 🡪 Cercare di “stare al passo” con le tecnologie e con le competenze richieste (avvicinandosi allo stato dell’arte, come detto poco fa)
  + “Imparare a imparare”, essenziale competenza trasversale, soprattutto tramite il

problem-solving, quindi adattarsi a rispondere ad esigenze specifiche di un certo tipo

L’obiettivo è cercare di avvicinarci a un modo di lavorare (way of working) professionale, cioè «operante allo stato dell’arte», per conoscenze tecnologiche e metodologiche, colmando quanto ci manca con il costante auto-apprendimento.

La conoscenza passa dalla comprensione profonda, sperimentata, dei significati di quanto si sta facendo

* Non ricordare, ma riconoscere (so chi sei …)

Vogliamo fissare tali conoscenze in un glossario

* Raccolta di termini/concetti centrali al dominio SWE (Software Engineering, da ora in avanti abbreviato; spesso, tali termini sono in inglese e si offre quando e dove utile, una traduzione, una definizione ed un chiarimento)
* Registrati in modo da facilitarne la localizzazione
* Corredati dalla nostra personale specifica del loro significato e ogni altra informazione utile a riconoscerli

Vogliamo raffinarne costantemente la comprensione

* Legando la teoria (quanto ascoltato) con la pratica (quanto riscontrato)

Questo nel nostro corso si realizza tramite un progetto didattico collaborativo (a scopo di learning, rispetto ad un futuro professionale)

* Promosso da un proponente esterno (aziende terze che portano un’idea)
* Con esigenze e obiettivi funzionali innovativi
* Complesso, impegnativo, visionario
* Tecnologicamente avanzato

Occorre fare attenzione alle norme e regole, per *non inabissarsi e sparire dai radar* (come un sottomarino e partire con un’idea, non riuscire a realizzarla perdendosi in pratiche inutili, buon paragone usato dal prof).

In generale (segue una definizione di Harold Kerzner, autore di “Project Management” ed esperto mondiale in materia) del termine:

Progetto: Insieme di attività che

* + - Devono raggiungere determinati obiettivi a partire da determinate specifiche (rispondendo ai *needs-bisogni*)
    - Hanno una data d’inizio e una data di fine fissate
    - Dispongono di risorse limitate (persone, tempo, denaro, strumenti)
    - Consumano tali risorse nel loro svolgersi

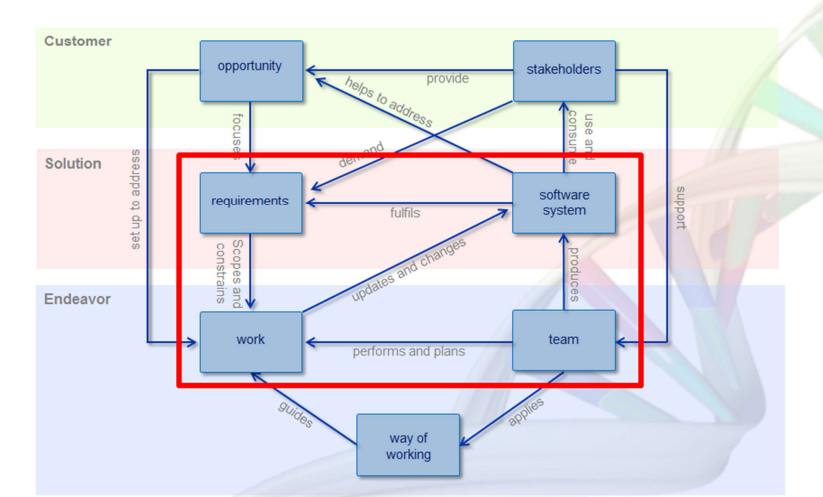
L’uscita di un progetto è un prodotto composito

* + SW (software) sorgente/eseguibile, librerie, documenti, manuali

La vera definizione in ambito universitario è *assignment*, quindi un compito specifico assegnato e gestito in maniera flessibile, sia di chi impone la scadenza, sia di chi lo esegue.

I costituenti di un progetto sono:

* Pianificazione 🡪 Studiare la natura del problema, capire come impostare le attività sulla base della domanda e pianificare l’organizzazione delle risorse
  + Gestire risorse (persone, tempo, denaro, strumenti) in modo responsabile, in funzione degli obiettivi
* Analisi dei requisiti 🡪 I needs sono tradotti in prodotti informatici (non sapendo come fare, si chiede e dialoga con una serie di interlocutori)
  + Definire cosa bisogna fare, normalmente compreso tramite interazione continuativa tra le parti in gioco (quelli che in project management sono chiamati *stakeholders*, quindi “portatori di interesse”)
* Progettazione (design) 🡪 Trasformare l’idea di comprensione (profonda) del problema in modo attuabile e con una soluzione sensata ed accettabile. Qui non si ha una sola soluzione, ma ce ne possono essere tante
  + Definire come farlo
* Realizzazione (implementation) 🡪 Creare un *utensìle* (cioè, avere un prodotto *usabile*; quindi, una volta creato compie correttamente la sua funzione e sia utile allo scopo voluto)
  + Farlo, perseguendo qualità (cioè, il grado di bontà oggettiva delle azioni eseguite)
  + Accertando l’assenza di errori od omissioni
  + Accertando che i risultati soddisfino le attese



Vedere la possibilità di creare

un prodotto utile allo scopo possibile

Figura - Proveniente da https://semat.org/

Chi ha cosa da dire significative sulle opportunità, sui bisogni e sulla soluzione/implementazione della stessa.

Possono essere clienti, committenti, venture capitalists (investitori)

Stato dell’arte, adattato al proprio livello sulla base delle regole presenti

Collassare design ed implementazione, esempio scrivendo codice, è tipico di chi progetta male.

Prima si pensa bene a cosa fare e l’implementazione risulta essere conseguenza (non premessa) del design.

In tal modo, mi assicuro di soddisfare le aspettative di chi lo richiede, chiedendo ed interagendo spesso.

Cosa non è un progetto:

* Si è accecati dalla fondamentale inutilità di fondo dei loro prodotti, dal senso di successo che si prova nel farli funzionare (il fatto che compili/funzioni ci toglie dall’esaminare ulteriormente un prodotto)
* In altre parole, i loro difetti fondamentali di progettazione sono completamente nascosti dai loro difetti di design difetti superficiali di progettazione (errori di sintassi, segmentation fault, ecc.)

Esso dovrebbe essere l’espressione di realizzare sulla base di regole un prodotto preciso e professionale, valorizzando l’esperienza e creandole al meglio delle regole esistenti.

L’opportunità è il progetto stesso; il bisogno c’è ed esiste e si cerca di anticipare la richiesta e soddisfare il bisogno.

Altri termini del glossario:

* Teamwork
  + Lavoro collaborativo che punta a raggiungere un obiettivo comune in modo efficace ed efficiente
  + I membri del team sono inter-dipendenti
  + La gestione di questa inter-dipendenza richiede il rispetto di regole e di buone pratiche
    - Comunicazioni aperte e trasparenti: risoluzione dei conflitti
    - Costruzione e preservazione delle fiducia reciproca: condivisione e collaborazione
    - Assunzione di responsabilità: coordinamento
    - Condivisione dei rischi
  + La sua base è un solido way of working
* Stakeholder (portatore di interesse)
  + Tutti coloro che a vario titolo hanno influenza sul prodotto e sul progetto
    - La comunità degli utenti (che usa il prodotto)
    - Il committente (che compra il prodotto)
    - Il fornitore (che sostiene i costi di realizzazione)
    - Eventuali regolatori (che verificano la qualità del lavoro)
* Way of working
  + Come organizzare al meglio le attività di progetto (in modo professionale)

Per svolgere un progetto potendo confidare nel suo successo serve ingegneria, quindi usando principi noti ed autorevoli (best practices), basato sulla software engineering (SWE), per garantire i migliori risultati in circostanze note e specifiche.

* Disciplina per la realizzazione di prodotti SW così impegnativi da richiedere il dispiego di attività collaborative
* Capacità di produrre “in grande” e “in piccolo”
* Garantendo qualità: efficacia (misura della capacità di raggiungere l'obiettivo prefissato)
* Contenendo il consumo di risorse: efficienza (misura dell'abilità di raggiungere l’obiettivo impiegando le risorse minime indispensabili)
* Lungo l’intero periodo di sviluppo e di uso del prodotto: ciclo di vita (Gli stati che il prodotto SW richiesto assume dal suo concepimento (bisogni 🡪 needs) all’uso e poi eventualmente al ritiro

Un sistema SW è tanto più utile quanto più è usato.

* Metrica: integrale della sua intensità d’uso nel tempo

Più lunga la vita d’uso di un prodotto, maggiore il suo costo di manutenzione

* Manutenzione: insieme di attività necessarie a garantire l’uso continuativo del prodotto
  + Reattivamente (per correzione dopo malfunzionamento) o preventivamente

Il costo di manutenzione ha varie componenti

* Mancato guadagno, perdita di reputazione, recupero o reclutamento esperti, sottrazione di risorse ad altre attività

I principi SWE puntano ad abbassare tali costi

* Sviluppando SW più facilmente manutenibile

L’obiettivo è raccogliere, organizzare, consolidare la conoscenza (body of knowledge) necessaria a realizzare progetti SW con efficacia ed efficienza (collezione e manutenzione migliorativa di best practice) e quindi applicare principi ingegneristici calati nella produzione del SW. Il modo di applicazione è:

* Sistematico
  + Modo di lavorare metodico e rigoroso
  + Che conosce, usa ed evolve le best practices di dominio
* Disciplinato
  + Che segue le regole che si è dato
* Quantificabile
  + Che permette di misurare l’efficienza e l’efficacia del suo agire

Distinguiamo infatti la figura del programmatore da quella del software engineer:

* Il programmatore scrive programmi in modo tecnico e personalizzato
* Il software engineer realizza parte di un sistema complesso con la consapevolezza che potrà essere usato, completato e modificato da altri. Egli comprende il contesto in cui si colloca il sistema cui contribuisce e cerca di attuare dei compromessi intelligenti e lungimiranti tra costi – qualità,

risorse – disponibilità, esperienza utente – facilità di realizzazione

Il metodo di acquisizione delle competenze avanza in corrispettivo alle sfide proposte, ciascuna con un punto di partenza e di arrivo, creandosi la condizione giusta per apprendere ed entrare nel flow (idealmente, Zone of Proximal Flow, quindi creandosi gli strumenti, apprendere e mettere in pratica nel concreto in maniera continuata e migliorante).

L’obiettivo è creare un *MVP (Minimum Viable Product)*, il prodotto fa quello che deve fare in un modo accettabile; chi lo decide è un accordo tra il committente e l’incaricato. Essa è una soglia minima e punta ad arrivare verso ciò che è fattibile.

06/10/2022: Processi di ciclo di vita

Un sistema SW, tanto più è usato, dovrebbe essere sottoposto sempre a manutenzione, la quale permette di risalire ad uno storico di vita di un SW. L’uso permette di scoprire necessità, difetti, adattandosi alle nuove esigenze e creandosi nuovi obiettivi.

Lo sviluppo di per sé ha un fine esecutivo (crei un software e finita lì); la manutenzione è il continuare in parte lo sviluppo, definendo un ciclo di vita per gli utenti che imparano a volerlo usare.

Gli stessi numeri di versione definiscono quanti cambiamenti: ovviamente, più grande è, più il software è cambiato. A noi interessa farla bene, dando una qualità crescente al prodotto, dando il giusto ordine alle operazioni (individuando il giusto e lo sbagliato in quanto fatto/eseguito).

Questa idea è il controllo di versione. Esso è un attributo decisivo in base alle esigenze dell’utente, che magari funzionano meglio in una versione precedente; ovviamente, occorre un giusto controllo tra le versioni, affinché non si perda tempo a sviluppare usando versioni sconnesse tra di loro.

Un software non è un oggetto statico; per esempio, un software scritto in un solo linguaggio per un solo programma non è manutenibile. Non è possibile rimettere insieme i pezzi dopo di un software (modularizzare); il software ben fatto è costituito di parti, le quali sono disposte secondo un preciso ordine.

L’idea chiave è che: se cambia il sorgente, allora cambia anche il prodotto. L’insieme delle parti del software dà luogo ad un prodotto stabile.   
Alla base, quindi, si cerca di compiere degli aggiustamenti a qualcosa già stabile di per sé.

La decisione di riconoscere quali parti usare e come tenerle insieme è detto configurazione.

Da ciò, consegue sia il citato controllo di versione, integrato anche con il controllo di configurazione.

Le parti che cambiano molto sono le parti molto esposte agli utenti e le parti che cambiano poco sono dei protocolli che rimangono fissi. La storia di un prodotto rappresenta delle maturazioni progressive, definite come stati. Le transizioni tra i singoli stati e le attività svolte sono gli archi.

Quali/quanti stati/regole attivino/abilitino gli archi dipende da:

* Vincoli/obblighi contrattuali, impegni (way of working), opportunità

*Altra definizione di progetto*:

Ogni progetto ha il compito di cercare di spingere un prodotto verso una nuova fase del suo ciclo di vita, con il solito obiettivo di creare qualcosa di usabile (implementation sulla base del design). In effetti, il ciclo di per sé intende l’usuale compiere attività ripetute comunemente; ciò può essere una conseguenza della manutenzione, quindi la ripetizione di attività iniziali.

Più nello specifico, il ciclo di vita del SW intende gli stati che il prodotto SW assume tra concepimento e ritiro in conseguenza delle attività svolte su di esso.

Il cambiamento di stato viene causato dalla comprensione dei *needs* e la progettazione del *design* sia per motivazioni interne che esterne allo stato di sviluppo. Ciò significa che:

* prendo un software non esistente, lavorando sulla base dei needs
* prendo un software esistente e cerco di mantenermi ai needs iniziali, qualificando il design realizzandolo progressivamente con l’implementation, mantenendo i principi cardine ed evolvendolo

Definiamo in generale:

* i *processi* di ciclo di vita, che raggruppano e codificano le transizioni nel ciclo di vita di un SW. In generale, sono insiemi di attività correlate e coese che trasformano ingressi (bisogni) in uscite (prodotti) secondo regole date e consumando risorse allo stesso tempo per farlo (definizione standard ISO 9000).
* i *modelli* del ciclo di vita, capendo quali stati/transizioni privilegiare, quali processi attivare ed attuare aderendo ad un modello e pianificare le attività, eseguendole e controllandole

Le regole dello stato dell’arte hanno stabilito quali attività privilegiare seguendo la logica del ciclo di vita, in cui ogni attività va opportunamente configurata.