1. Grok

1. INTRODUZIONE (T01)

1.1 Frasi da dire all'orale (obbligatorie)

> "L'obiettivo del corso è apprendere un way of working professionale, cioè lavorare allo stato dell'arte dell'ingegneria del software, perché un sistema software è tanto più utile quanto più è usato, e più è usato più deve essere mantenuto. La manutenzione è la fase più lunga e costosa del ciclo di vita (70-90% del costo totale), quindi l'ingegneria del software serve a produrre software facilmente mantenibile."

1.2 What – Why – How (schema da sapere a memoria)

Domanda	Risposta esatta (da dire così)
What	Apprendiamo metodi e pratiche professionali: gestire il tempo, collaborare, assumersi responsabilità, auto-aprenderci
Why	Perché il software vive decenni → manutenzione domina i costi → SWE = riduzione drastica del costo di manutenzione
How	Progetto didattico di gruppo complesso + esame scritto/orale

1.3 Soft skills richieste (domanda classica)

1. Problem-solving skills
2. Acting as a team player (non individualismo) ← sottolineato 3 volte nelle slide
3. Time management abilities
4. Ability to accept and learn from criticism
5. Flexibility/adaptability
6. Working well under pressure
7. Strong work ethic, Positive attitude

1.4 Definizioni fondamentali (impara a memoria)

Termine	Definizione esatta (citazione d'esame)
Progetto (Kerzner)	"Un insieme di attività che devono raggiungere obiettivi specifici, con data di inizio e fine fissata, con risorse limitate e che consumano risorse"

Termine	Definizione esatta (citazione d’esame)
Way of working	L’insieme delle pratiche, processi, strumenti e comportamenti che un team usa per operare allo stato dell’arte
Stakeholder	Chiunque abbia interesse o influenza sul prodotto o sul progetto
Manutenibilità	Facilità con cui un sistema può essere modificato (correttiva, adattativa, perfettiva, preventiva)

---

2. PROCESSI DI CICLO DI VITA (T02)

2.1 Perché esiste la manutenzione → perché esiste il ciclo di vita

1. Un sistema SW è utile ∝ tempo di utilizzo
2. Più è usato → più deve essere mantenuto
3. Tutto ciò che è sotto manutenzione ha una **storia**
   → serve **controllo di versione** (non perdere nulla, poter tornare indietro)
   → serve **controllo di configurazione** (il prodotto non è monolitico)

2.2 Classificazione processi ISO/IEC 12207 (da sapere a memoria)

Tipo	Processi (esempi principali)	Note
Primari	Acquisizione, Fornitura, **Sviluppo**, Operatività, **Manutenzione**	Un progetto esiste solo se attiva almeno uno di questi
Di supporto	Documentazione, Gestione configurazione, Assurance qualità, **Verifica**, **Validazione**, Revisione	Come “funzioni” rispetto al main
Organizzativi	Gestione processi, Infrastruttura, Miglioramento, Formazione	Trasversali

2.3 Definizioni chiave (domande frequenti)

Termine	Definizione (da dire così)
Processo	Insieme di attività correlate che trasformano ingressi in uscite consumando risorse
Efficienza	Fare le cose senza sprecare risorse
Efficacia	Raggiungere gli obiettivi
Economicità	**Efficienza + Efficacia** → obiettivo di ogni attività di progetto

Termine	Definizione (da dire così)
PDCA	Plan-Do-Check-Act → ciclo di **miglioramento continuo** del way of working (NON è il ciclo di vita del prodotto!)

---

3. IL CICLO DI VITA DEL SOFTWARE (T03)

3.1 Concetto fondamentale (frase d'oro)

> "Il ciclo di vita è la sequenza di stati che un prodotto software assume dalla concezione al ritiro. A noi interessa solo il segmento [concezione → sviluppo]. La transizione tra stati avviene per azione di processi di ciclo di vita. L'obiettivo di un progetto è far progredire lo stato di avanzamento del prodotto mobilita specifiche attività di processi, ordinate secondo le dipendenze tra ingressi e uscite, fissando pre-condizioni (quando iniziare) e post-condizioni (quando finire)."

3.2 Modelli di ciclo di vita (tabella perfetta per l'esame)

Modello	Caratteristiche principali	Quando usarlo	Difetto principale
Code-and-Fix	Scrivi → correggi → ripeti	Prototipi molto piccoli	Caotico, non scalabile
Cascata	Fasi sequenziali rigide, gate reviews	Requisiti stabili, progetto piccolo	Nessun ritorno indietro
Incrementale	Più rilasci successivi, requisiti ordinati per importanza	Feedback precoce importante	Architettura deve essere definita presto
Iterativo (Spirale)	Cicli di raffinamento + gestione esplicita del rischio	Alta incertezza	Rischio di non convergenza
Agile (Scrum/XP)	Manifesto 2001: individui > processi, SW funzionante > documentazione	Requisiti volatili, cliente presente	Richiede disciplina ferrea

3.3 SEMAT Essence – Kernel (domanda quasi sicura)

7 aree di preoccupazione (alpha):

1. Opportunity
2. Stakeholders
3. Requirements
4. Software System
5. Work

6. Team
7. Way of Working

Stati tipici (esempio Requirements):
Conceived → Bounded → Coherent → Acceptable → Addressed → Fulfilled

---

4. GESTIONE DI PROGETTO (T04)

4.1 Definizione di progetto (da recitare parola per parola)

> "Un progetto è un **insieme ordinato di attività istanziate da processi di ciclo di vita**.
> Le attività sono composte da compiti assegnati a singoli individui.
> Il progetto è **sempre collaborativo**.
> Le attività sono **pianificate prima** di essere svolte.
> Ogni attività ha obiettivi e vincoli derivanti dal processo di appartenenza e deve perseguire **economicità**."

4.2 Passi della pianificazione (WBS + scheduling)

1. Identificare attività (Work Breakdown Structure – gerarchica)
2. Identificare dipendenze (logiche/tecnologiche)
3. Stimare effort e durata
4. Allocare persone e risorse
5. Produrre diagrammi (Gantt, PERT)

4.3 Stima dei costi (metodi + fattori COCOMO II)

Metodi: analogia, expert judgment, algoritmi (COCOMO II)
17 fattori di costo (es. complessità, esperienza team, vincoli di tempo…)

4.4 Gestione del rischio (ciclo continuo – da sapere a memoria)

1. Identificazione
2. Analisi qualitativa (probabilità × impatto → matrice 5×5)
3. Analisi quantitativa (EMV = P × I)
4. Pianificazione risposta (evitare, mitigare, trasferire, accettare)
5. Monitoraggio e controllo

4.5 Standish Group Chaos Report (citazione obbligatoria)

Anno	Successo	Falliti	Challenged
1994	16%	31%	53%

Anno	Successo	Falliti	Challenged
2004	34%	15%	51%

→ Il miglioramento è dovuto a **migliori pratiche di gestione progetto**.

---

5. ANALISI DEI REQUISITI (T05) – ARGOMENTO PIÙ PESANTE

5.1 Definizione IEEE di “requisito” (da sapere a memoria)

1. **Lato bisogno**: capacità necessaria a un utente per risolvere un problema o raggiungere un obiettivo
2. **Lato soluzione**: capacità che un sistema deve avere per soddisfare contratto/standard/specifica

Documento	Contenuto	Punto di vista
Capitolato	Requisiti utente	Lato bisogno
Specifica requisiti	Requisiti software	Lato soluzione

5.2 Proprietà di una buona specifica (elenco completo)

- Completa
- Coerente
- Non ambigua
- **Verificabile** (esiste un test finito e oggettivo)
- **Tracciabile**
- Modificabile
- Prioritizzata (MoSCoW)

5.3 Tracciabilità (concetto centrale)

- Matrice requisito utente ↔ requisito software
- Garantisce **necessità** (ogni req SW deriva da almeno un req utente) e **sufficienza** (tutti i bisogni sono coperti)

5.4 Verifica vs Validazione (domanda classica – frase perfetta)

> “**Verifica**: ‘Sto costruendo il prodotto giusto?’ → confronto con la specifica (ispezioni, review, test di unità/integrazione)
> **Validazione**: ‘Sto costruendo il prodotto che serve?’ → confronto con le aspettative degli stakeholder (prototipi, collaudo, UAT)”

5.5 Stati SEMAT Requirements

Conceived → Bounded → Coherent → Acceptable → Addressed → Fulfilled

---

6. PROGETTAZIONE SOFTWARE (T06)

6.1 Modello a V (DISEGNO OBBLIGATORIO ALL'ESAME)

```
                Validazione
                   ↑
Capitolato → Analisi req. → Progettazione logica → Progettazione dettaglio →
Codifica → Collaudo
                   ↑               ↑                    ↑
↑
                Verifica          Verifica             Verifica
Verifica
```

6.2 Obiettivo della progettazione

> "Dominare la complessità attraverso decomposizione, information hiding, alta coesione, basso accoppiamento."

6.3 Principi cardine (da citare sempre)

Principio	Autore/Anno	Frase d'esame
Information Hiding	Parnas 1972	"Ogni modulo deve nascondere un segreto di progettazione"
Alta coesione	Yourdon/Constantine	"Elementi che collaborano allo stesso scopo devono stare insieme"
Basso accoppiamento		"Minimizzare le dipendenze tra moduli"
Rasoio di Occam		"La soluzione più semplice possibile, ma non più semplice di così"

6.4 Qualità del design (elenco)

- Sufficienza, comprensibilità, modularità, robustezza, flessibilità, riusabilità, efficienza, affidabilità, sicurezza, semplicità, incapsulamento

6.5 Design Patterns (GoF 1995) – categorie

- **Creazionali**: Factory Method, Singleton, Builder
- **Strutturali**: Adapter, Decorator, Facade, Proxy

- **Comportamentali**: Observer, Strategy, Command, State, Template Method

---

Questo documento costituisce la revisione, più dettagliata e precisa, della sintesi richiesta. Essa è stata strutturata per operare come una **Guida allo Studio di Ingegneria del Software (SWE)** di livello universitario triennale, applicando i principi di **disciplined reasoning** e **technical creativity** richiesti per l'analisi dei contenuti forniti.

La trattazione è segmentata secondo una progressione logica, dalle fondamenta disciplinari ai modelli di sviluppo e ai processi critici di specificazione e soluzione.

---

2. Gemini

I. Fondamenti, Definizione e Ruolo Disciplinare della SWE

1. Definizione e Ambito (SWE)

L'**Ingegneria del Software (SWE)** è definita come l'applicazione di un approccio **sistematico, disciplinato e quantificabile** allo sviluppo, esercizio e manutenzione del software, includendo lo studio di tali approcci. L'obiettivo primario non è la semplice *scrittura di codice* (Programming), ma la **gestione della complessità** e la garanzia della **qualità** (Affidabilità, Manutenibilità, Efficienza) e dell'**economicità** del prodotto nel lungo termine.

- **Distinzione Programmator vs. Ingegnere del Software:**
  - **Programmatore:** Concentra l'attività sulla creatività tecnica (*Problem Solving* a livello di implementazione), spesso lavorando in autonomia o su porzioni limitate, con focus primario sul codice funzionante.
  - **Software Engineer:** Opera in un **Team** (Teamwork), applica **metodi e pratiche** standardizzate (best practice), gestisce il progresso per obiettivi e opera sotto **vincoli temporali e di risorsa** specifici. La sua responsabilità è sia tecnica che gestionale.

2. Il Concetto di Progetto

Un **Progetto** è l'insieme ordinato e collaborativo di attività con un obiettivo ben definito, un inizio e una fine temporale stabiliti, e specifiche risorse allocate. Il successo del progetto è intrinsecamente legato alla gestione efficace dei **processi di ciclo di vita** che vengono *istantiati* in specifiche attività progettuali.

II. Processi di Ciclo di Vita del Software e Standardizzazione

I **Processi di Ciclo di Vita** gestiscono le transizioni tra gli stati del prodotto (dalla concezione al ritiro/disattivazione). Lo standard di riferimento fondamentale è **ISO/IEC 12207** (o gli standard successivi come ISO/IEC/IEEE 15288 per i sistemi).

1. Classificazione dei Processi (ISO/IEC)

Lo standard classifica i processi in tre categorie principali:

Categoria	Scopo Operativo e Rilevanza	Esempi di Processi Chiave
Primari	Riguardano la **trasformazione diretta** del prodotto software, dal bisogno utente al sistema operativo. Definizione del *cosa* e del *come*.	**Acquisizione** (Stakeholder), **Fornitura** (Fornitore), **Sviluppo** (Design, Implementazione), **Manutenzione** (Correttiva, Adattiva, Perfettiva).
di Supporto	Aumentano la **qualità** e la **tracciabilità** delle attività primarie, garantendo che il prodotto sia *corretto* e *conforme*.	**Gestione della Configurazione** (Versionamento), **Assicurazione della Qualità (QA)**, **Gestione della Documentazione**, **Verifica** e **Validazione**.
Organizzativi	Riguardano il **miglioramento continuo** del *way of working* dell'organizzazione e del suo ambiente. Non agiscono direttamente sul prodotto.	Gestione dell'Infrastruttura, Formazione, **Miglioramento del Processo**.

2. Differenziazione Critica (Verifica vs. Validazione)

I processi di supporto di **Verifica (Verification)** e **Validazione (Validation)** sono distinti e complementari:

- **Verifica (Did we build the product *right*?):** Accertamento che i prodotti del lavoro (es. documentazione, codice) siano conformi ai requisiti specificati per quella fase. Spesso svolta tramite ispezioni, *walkthrough* e test di unità/integrazione.
- **Validazione (Did we build the *right* product?):** Accertamento che il software soddisfi l'uso previsto dall'utente e i bisogni originali. Si focalizza sul punto di vista dell'utente finale.

3. Miglioramento dei Processi (Ciclo PDCA)

I processi organizzativi di miglioramento seguono il ciclo **PDCA (Plan, Do, Check, Act)**.

- **Ambito:** Il ciclo PDCA **non** corrisponde alla normale sequenza di attività di progetto e **non** opera sul prodotto, ma sui *processi* per ottimizzarne l'efficacia (es. ridurre i difetti, aumentare l'efficienza).

III. Modelli di Ciclo di Vita del SW e Dinamiche di Sviluppo

I **Modelli di Ciclo di Vita** definiscono la sequenza delle fasi e le transizioni ammesse (i processi attivati).

1. Modello Sequenziale (Waterfall)

- **Struttura:** Fasi rigidamente sequenziali (Requisiti $\rightarrow$ Design $\rightarrow$ Implementazione $\rightarrow$ Test $\rightarrow$ Manutenzione). Il *Ritorno* è formalmente ammesso solo in fase di Manutenzione.
- **Vantaggi/Svantaggi:** Facile da gestire per progetti con requisiti stabili e ben definiti. Altamente inefficiente per sistemi complessi o nuovi, in cui i requisiti evolvono (alto rischio di scoperte tardive e costi di *rework* elevati).

2. Dinamiche Iterative e Incrementali

Questi modelli (prototipazione, a spirale, *Agile*) sono l'evoluzione logica per la gestione della complessità e del cambiamento (Collapso Tecnologico, Mutamento Requisiti, ecc.).

- **Iterazione:** Attività di raffinamento, riscrittura (*refactoring*) o rivisitazione di una porzione esistente del sistema. Può avere un effetto costruttivo o distruttivo sul prodotto, ma mira al miglioramento della sua struttura o funzionalità.
- **Incremento:** Aggiunta di una nuova funzionalità funzionante e verificata a un impianto base stabile (*baseline*). Ha un effetto sempre **costruttivo** sull'insieme.
- **Mitigazione del Rischio:** La scomposizione del sistema e l'uso di cicli brevi e controllati (iterazioni/sprint) riducono il rischio di non convergenza o di insuccesso totale, distribuendo le verifiche e le validazioni nel tempo.

3. Metodi Agili (Agile)

I metodi Agili (come Scrum, XP) nascono per massimizzare la capacità di **rispondere al cambiamento**.

- **Manifesto Agile:** Fissa quattro valori fondamentali che preferiscono:
  1. **Individui e interazioni** rispetto a processi e strumenti.
  2. **Software funzionante** rispetto a documentazione esaustiva.
  3. **Collaborazione col cliente** rispetto a negoziazione contrattuale.
  4. **Rispondere al cambiamento** rispetto a seguire un piano.

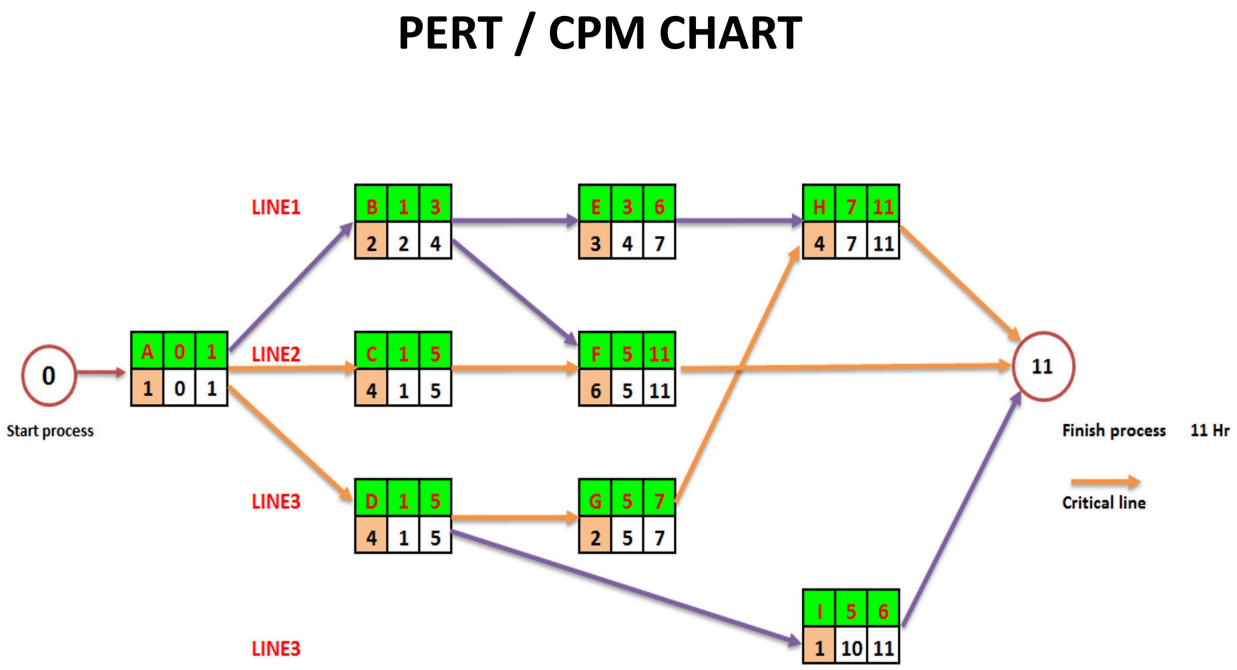
IV. Gestione di Progetto e Analisi dei Rischi (Project Management)

La gestione di progetto è l'applicazione di conoscenze, abilità, strumenti e tecniche alle attività di progetto per soddisfare i requisiti.

1. Strumenti di Pianificazione e Tempo

La pianificazione è la fase di definizione delle attività, delle dipendenze, delle risorse e delle scadenze (milestone).

- **Diagramma di Gantt:** Strumento grafico per la rappresentazione temporale delle attività, utile per il monitoraggio dello stato di avanzamento.
- **Tecnica PERT (Programme Evaluation and Review Technique):** Usata per analizzare e rappresentare le dipendenze tra attività in un grafo orientato. L'obiettivo primario è determinare il **Cammino Critico (Critical Path)**.
  - **Cammino Critico:** La sequenza di attività, dalla prima all'ultima, la cui durata aggregata determina la durata minima possibile dell'intero progetto. Qualsiasi ritardo su un'attività del cammino critico si traduce in un ritardo equivalente nella data di fine progetto.



Shutterstock

2. Analisi e Mitigazione dei Rischi

Il processo di gestione dei rischi è cruciale. Un **Rischio** è un evento incerto che, se si verifica, ha un effetto positivo o negativo sugli obiettivi del progetto.

- **Fasi del Risk Management: Identificazione** (cosa può andare storto), **Analisi** (quantificazione di probabilità e impatto/conseguenza), **Pianificazione della Risposta**

(mitigazione, evitamento, trasferimento, accettazione), **Controllo** (monitoraggio e attuazione dei piani di emergenza).

- **Fattori Critici di Fallimento (Standish Group):** Le principali cause di fallimento dei progetti software sono storicamente legate alla fase iniziale di specificazione e gestione del cliente:
  1. **Requisiti incompleti** (13.1%)
  2. **Mancato coinvolgimento del cliente** (12.4%)
  3. **Aspettative non realistiche** (9.9%).

V. Processi di Specificazione e Soluzione (Requisiti e Design)

1. Ingegneria dei Requisiti (Requirements Engineering)

L'attività di analisi dei requisiti risponde al quesito **"Qual è il problema, qual è la cosa giusta da fare?"**.

- **Definizione Dicotomica del Requisito (IEEE):**
  - **Lato Bisogno (Requisito Utente):** Capacità richiesta dall'utente (espressa nel **Capitolato**).
  - **Lato Soluzione (Requisito Software):** Capacità richiesta al sistema per soddisfare il bisogno (espressa nella **Software Requirements Specification - SRS**).
- **Tracciabilità (Traceability):** È l'elemento di connessione logica che garantisce che l'insieme dei requisiti software sia **Necessario** (ogni requisito SW serve a soddisfare un bisogno utente) e **Sufficiente** (tutti i bisogni utente essenziali sono coperti dai requisiti SW).
- **Qualità della SRS:** La documentazione deve essere **non ambigua, completa, consistente, tracciabile e verificabile**.

2. Progettazione Software (Software Design)

L'attività di progettazione (Design) risponde al quesito **"Come si costruisce il prodotto?"** e definisce la soluzione.

A. Il Modello a V e i Livelli di Design

Il **Modello a V** illustra la mappatura tra le fasi di produzione (sinistra) e le corrispondenti attività di verifica (destra).

Fase di Produzione (Sinistra)	**Corrispondente Fase di Verifica (Destra)**
Analisi dei Requisiti	**Validazione** (Collaudo/Acceptance Test)
Progettazione Logica/Architetturale	**Verifica** (Test di Sistema)

Fase di Produzione (Sinistra)	Corrispondente Fase di Verifica (Destra)
Progettazione di Dettaglio	**Verifica** (Test di Integrazione)
Codifica	**Verifica** (Test di Unità)

B. Progettazione Logica (Architettura) vs. Dettaglio

1. **Progettazione Logica/Architetturale:** Definisce la **struttura** di alto livello: la **decomposizione** del sistema in componenti, i loro **ruoli**, le **interfacce** e le **interazioni**. La scelta architetturale deve massimizzare gli attributi di qualità extra-funzionale (affidabilità, flessibilità, manutenibilità).
2. **Progettazione di Dettaglio:** Specifica l'implementazione interna delle singole unità architetturali, incluse le strutture dati locali e gli algoritmi.

C. Principi Cardine della Modularità (Qualità)

La qualità del design è determinata dall'efficacia con cui si gestiscono le dipendenze tra moduli/componenti:

- **Coesione (Cohesion):** Misura il grado in cui gli elementi interni di un modulo sono correlati per raggiungere un unico scopo funzionale.
  - **Principio:** Si deve tendere a **Alta Coesione** (un modulo deve fare *una* cosa, farla bene, e tutti i suoi elementi devono essere necessari per quello scopo).
- **Accoppiamento (Coupling):** Misura il grado di interdipendenza tra moduli diversi. Un accoppiamento elevato implica che una modifica in un modulo richieda spesso modifiche negli altri.
  - **Principio:** Si deve tendere a **Basso Accoppiamento** (le componenti devono interagire solo tramite interfacce ben definite e minimamente esposte).
- **Incapsulamento (Encapsulation):** Tecnica fondamentale per raggiungere un basso accoppiamento. Consiste nel nascondere l'implementazione e i dati interni di una componente, esponendo solo i servizi necessari (interfaccia pubblica), riducendo così le assunzioni esterne sul funzionamento interno (principio della *Black Box*).