CORRECT BY CONSTRUCTION :

→ possiamo partire con buoni esempi

DESIGN :

bisogno personale e poi specifico (difficile dividerlo subito)
(codifica)

ARCHITETTURA :

problema della progettazione

cosa fa? cosa dice? componenti

- **decomposizione in parti che si possono ricomporre**

  non succede quando ci manca qualche pezzo se la decomposizione è fatta male

  bene decomporre con metodo!

- **Organizzare le componenti**

  sapere dove devono mettere le componenti, ogni parte ha un ruolo nell'insieme, devono parlarsi.

  Una parte dice cosa offre, l'altra di cosa ha bisogno

- **Le componenti hanno delle relazioni**
- **Si ricompongono con uno stile** di buona proprietà

→ ogni parte deve sapere cosa occorre (la persona più importante in AUTONOMIA)

PASSI DI DESIGN :

- ARCHITETTURA LOGICA
- ARCHITETTURA DI MESSAGGIO (diritto all'implementazione)

quando dividato non c'è beneficio

↳ assegnabile a 1 persona, comprensibile a tutti

UNITÀ ARCHITETTURALI : la singola scatola che (↳ con interfacce)

- espongono come API (espongo cosa)
- chiedono come API (ho bisogno di)

↓ passa da parti + piccole

i c.d. programmi modularizzano:

C ha procedure

Java ha classi

l'implementazione la verifico con il design!

↓

deve rispecchiare l'unità architetturale

↓

compiti non devono essere uguali! altrimenti c'è attesa

↓

come misuro l'implementazione? (passa da smatman ↳ chiuse da ".")

↓

UNIT VERIFICATO : si può dimostrare!

DESIGN :

↓ tantissimo refactory

1) TOP-DOWN → decompongo (scompattare)
2) BOTTOM-UP → OOP (creatore ↓ conosco gli ingredienti)
3) AGILE

(compilatore, lexers)

OOP → classi istanziabili con costruttori, sono specializzabili

utile quando prolifera

DESIGN PATTERN :

non si possono ignorare

CODIFICA SENZA CONSEGUENZA PROGETTAZIONE

ACCOPPIAMENTO : individuare parti che hanno dipendenze

↳ primo obiettivo : confini

↳ cosa faccio (cosa espongo)

↳ cosa chiedo

cosa significa dire cosa è un modulo?

come suddivido in moduli?

- Pipeline (diviso in fasi, ognuna la segue di cose che facciamo) → esposto ai cambiamenti
- Information hiding (nascondo dettagli implementativi)

↳ dà meno problemi nei cambiamenti

↳ interfacce esposte

(de scopo decomponendo in componendo)

DISPONIBILITÀ : il sistema non è disponibile per un lasso di tempo

↓

correzione : ← sospendo il vecchio che non funziona, uso il nuovo (il sistema è tutto o niente!)

parti che possono funzionare da sole (sistema distribuito) (Internet)

SEMPLICITÀ : un modulo in più serve? se sì, perché?

INCAPSULAZIONE : operazioni accedute tramite interfacce

↳ internamente aggregate e nascoste all'esterno

↳ isolamento : cambiamenti

COESIONE : proprietà misurata in questo modo:

quello che c'è è tutto necessario?

(se togliamo qualcosa non funziona oppure qualcosa non c'entra)

↳ dà un conto sull'incapsulazione (è buona?)

↳ S. O. L. I. D. per misurare la coesione

↳ tipi di coesione (come misuro la coesione?)

- funzionale (rispetto agli scopi)
- temporale (rispetto alla sequenza temporale)
- informativa (i dati su cui lavoro) → forma + solida

ACCOPPIAMENTO : misura le dipendenze indesiderabili (ho decomposto male, ho incapsulazioni che non separano)

↳ fa danno nel rapporto tra interno ed esterno (ricerca assenza da fuori verso dentro!)

↳ come lo misuro? (FAN-IN, FAN-OUT)

- conto le dipendenze
- se è utile questo è utile una parte architetturale

cosa descrive l'architettura?

↓

uno strumento che conosce il linguaggio dei diagrammi (come un editor che conosce il linguaggio)

è studiabile rispetto alla sintassi

↳ ti dice quale componente non è usato o poco usato

NECESSITÀ ARCHITETTURALI : tornano sull'architettura
(SCRAP)

↳ 4 GRADI :

- **ARCHITETTURA INDIVIDUATA** : individuate le componenti

  2 approcci
  - prendi i pezzi che esistono
  - li fai

- **DIMOSTRABILE** : PoC

- **USABILE** : fa il suo mestiere (verso il PB)

  assomiglia al prodotto (ha le caratteristiche desiderate)

- **READY** : disponibile

RAPPORTO PoC - Architettura : poco design (le tecnologie essenziali)

PB : prodotto completo