REQUISITI SUCCINTI (EFFICACIA)

VERIFICAZIONE :

- <u>modo statico</u> : attività di verifica il cui oggetto non deve eseguire
- <u>modo dinamico</u> : sul programma in esecuzione

↳ **REQUISITI FUNZIONALI** : rapporto I - O

↳ **REQUISITI NON FUNZIONALI** : progettazione, caratteristiche che permettono di dire se è fatto bene

PROPRIETÀ :

- <u>ARCHITETTURA</u>
- <u>DI USO</u> (DIVERSA x OGNI UTENTE)
-

ADS scrivono molte altre fonti rispetto al capitolato

REQUISITI : proprietà (per noi; nostro) da soddisfare e da dimostrare

INTEGRITY : fa quello che deve fare e non deraglia
((la precisione di un algoritmo cambia? è intenzionale)
↳ quando ci si poggia su librerie esterne

facilita la verifica

PROGRAMMI ELEGANTI : non belli, sono leggibili, capibili e capire se assolvono le aspettative
↓
fare verifica all'indietro (cioè farla molto dopo)
è da evitare !! costo poi chiaramente + alto
↳
input pericolosi

fai tanto codice e subito
↑

AIUTARE LA VERIFICA :

- RIMUOVO PROBLEMI : spero che vada bene (poco costoso)
- (approccio pro attivo) PER COSTRUZIONE : enfatizza la verifica
  ↳ codice costantemente

DISCIPLINA PER PROGRAMMARE :

1. software con comportamento predicibile (non indovinare)
2. buoni principi di programmazione
3. atteggiamento pragmatico (parziale, non opinioni)

(1.) se lo leggo capisco che fa (WYSIWYG) (non eseguendolo), evito effetti collaterali:
↳ attenti all'elaborazione e inizializzazione (diversi x ogni O.S.)

(2.)
1. programmazione deve riflettere design
2. interfaccia / implementazione
3. information hiding
4. tipi specializzati per dati

(3.) software facilmente verificabile (prima di entrare in repo) (con commenti)

TRACCIAMENTO (fra prodotto e requisiti)
↳ ogni requisito è soddisfatto con un filo (design ... codice)
desiderabile che i requisiti siano assegnabili a un frammento di codice (decidibili)
↓
ATOMIZZARE I REQUISITI (un test → un requisito)
(NON DI 2)

ANALISI STATICA DEL CODICE

ci si serve a partire dal main