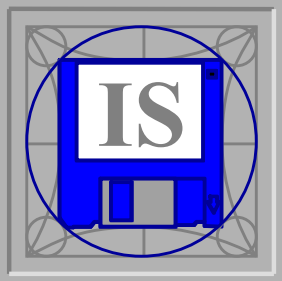
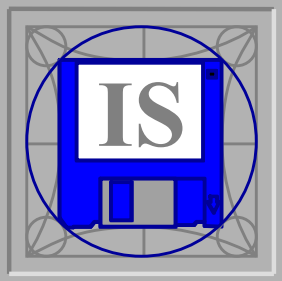


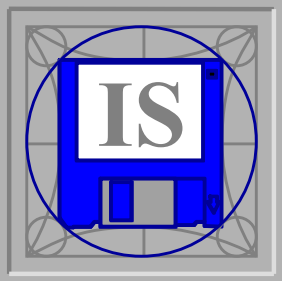
Appendice: qualità di progettazione

- Non tutti i problemi hanno una (buona) soluzione
- Per ogni scelta progettuale serve fissare con la massima chiarezza possibile
  - Obiettivi (della scelta)
  - Vincoli (nella scelta)
  - Alternative (alla scelta)
  - Come la soluzione corrisponde al problema
- Fattibilità e verificabilità sono qualità cardine della progettazione



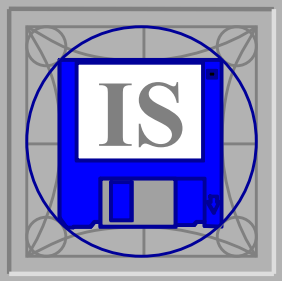
Buone tecniche di progettazione – 1/3

- **Decomposizione modulare**
  - Una buona decomposizione architetturale identifica componenti tra loro indipendenti
  - Minimo accoppiamento
  - Autosufficienza (coesione funzionale)
- **Incapsulazione (*information hiding*)**
  - Nascondere il dettaglio realizzativo
  - Solo l'interfaccia è pubblica
  - Il dettaglio è noto solo all'interno



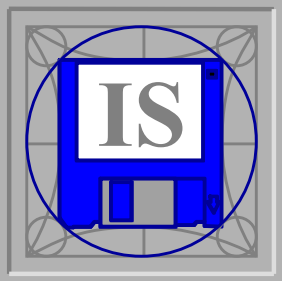
Buone tecniche di progettazione – 2/3

- Controllo di accoppiamento e di coesione
- L'accoppiamento è indicatore dell'intensità di relazione tra parti distinte
  - La modifica di una comporta modifiche nell'altra
  - Forte accoppiamento → cattiva modularità
- La coesione è indicatore dell'intensità di relazione all'interno di una singola parte
  - Forte coesione → buona modularità



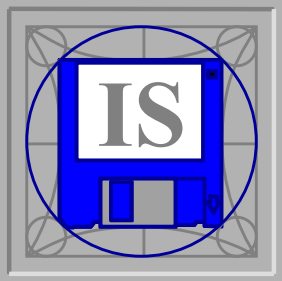
Buone tecniche di progettazione – 3/3

- L'astrazione omette informazione per poter applicare operazioni simili a entità diverse
  - Ciò che è caratteristico dell'intera gerarchia è fissato in radice
  - Ciò che differenzia si aggiunge per specializzazione allontanandosi dalla radice
- A ogni astrazione corrisponde una concretizzazione
  - Per parametrizzazione (p.es.: da *template* a entità concreta in C++)
  - Per specializzazione (p.es.: da interfaccia a classe in Java e C++)
  - Per valorizzazione (p.es.: da classe a oggetto tramite costruttore)



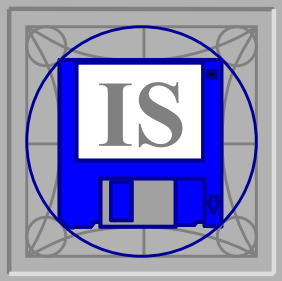
Problematiche critiche – 1/2

- La concorrenza di buona qualità aiuta a decomporre il sistema in più entità autonome garantendo
  - Efficienza di esecuzione
  - Atomicità di azione
  - Consistenza e integrità dei dati condivisi
  - Semantica precisa di comunicazione e serializzazione
  - Predicibilità di ordinamento temporale
- La distribuzione di buona qualità ripartisce il sistema in programmi disseminati su nodi distinti garantendo
  - Bilanciamento di carico
  - Frugalità nelle comunicazioni: buon grado di indipendenza



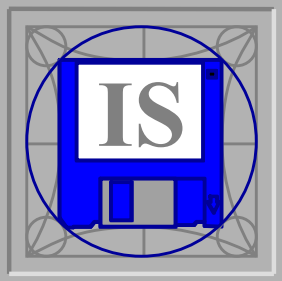
Problematiche critiche – 2/2

- Eventi ed errori in presenza di concorrenza o distribuzione
- In relazione al flusso dei dati
  - Saper determinare quando un certo dato è disponibile
- In relazione al flusso di controllo
  - Saper determinare l'ingresso in un particolare stato (dell'intero sistema o di una sua parte)
- In relazione al trascorrere del tempo
  - Saper determinare l'arrivo di certo istante temporale
- Mai fare assunzioni ottimistiche!



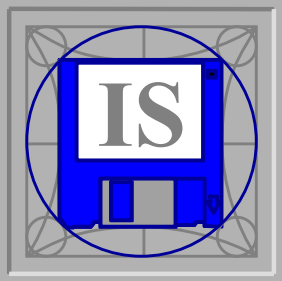
Ricercare integrità concettuale

- Facilmente riconoscibile nelle architetture fisiche
  - Suggerisce adesione a uno stile uniforme
  - Coerentemente in tutte le parti del sistema e nelle loro interazioni
- Bilancia ricchezza funzionale con semplicità d'uso
- Desiderabile in ogni architettura di sistema
  - Anche nel *software*
- Richiede osservanza e vigilanza
  - Facilita parallelismo nella realizzazione



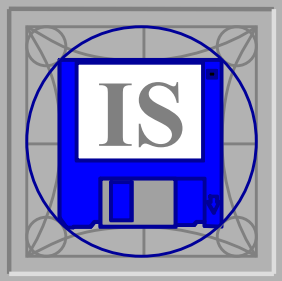
Consigli: *enforce intentions*

- Nel passaggio da progettazione a codifica
- Rendere chiaro il confine tra esterno e interno dei moduli
- Decidere chiaramente e codificare coerentemente ciò che può essere specializzato
  - Rendere il resto immodificabile (`final`, `const`, …)
- Proteggere tutto ciò che non deve essere visto e acceduto dall'esterno
  - `Private`, `protected`, …
- Decidere quali classi possono produrre istanze e quali no
  - Usare il *pattern* Singleton per le classi a istanza singola



Consigli: *defensive programming*

- Programmare esplicitamente il trattamento dei possibili errori
  - Nei dati in ingresso: verificarne la legalità prima di usarli
  - Nella logica funzionale: fissare proprietà (invarianti, pre- e post-condizioni, …)
- Definire la strategia di trattamento degli errori (*error handling*) è compito della progettazione



Gestire gli errori nei dati in ingresso

- Possibili tecniche di trattamento
  - Attendere fino all'arrivo di un valore legale
  - Assegnare un valore predefinito (*default*)
  - Usare un valore precedente
  - Registrare l'errore in un *log* persistente
  - Sollevare una eccezione gestita
  - Abbandonare il programma