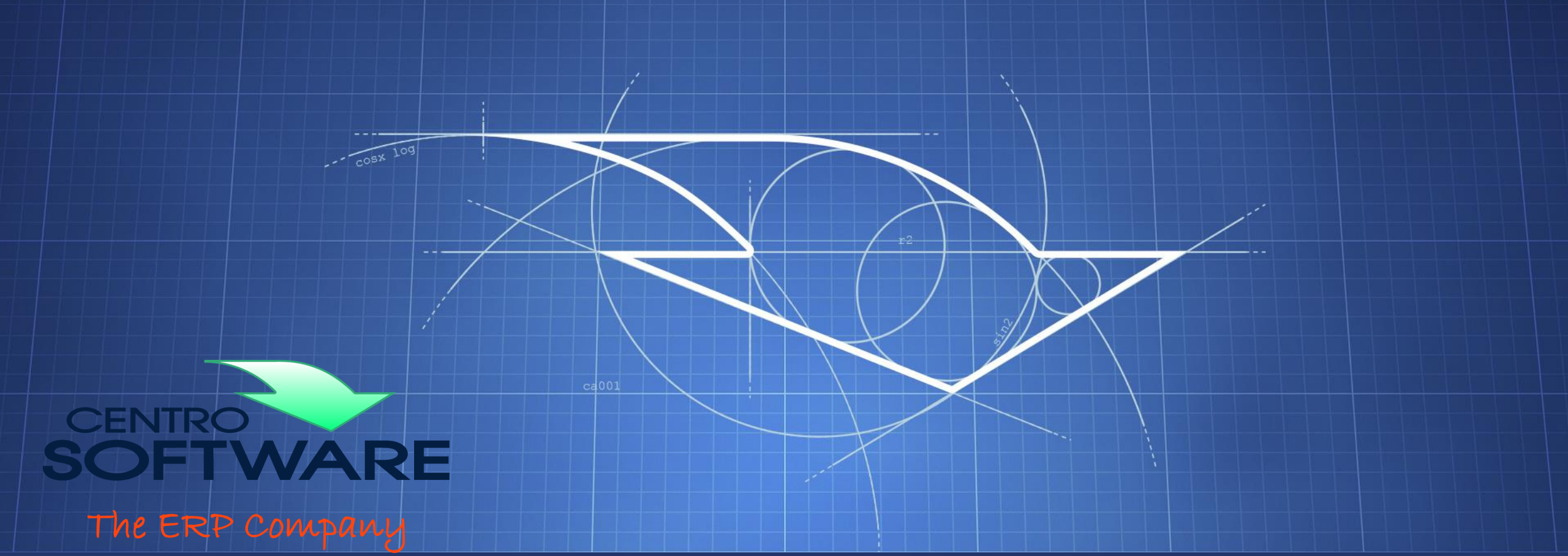


CENTRO SOFTWARE

The ERP Company

Sviluppo di applicazioni Enterprise

11 Dicembre 2017 – Sergio Govoni

Agenda

- Centro Software – The ERP Company

- Lo sviluppo di applicazioni Enterprise
  - La strategia
  - Gli ingredienti fondamentali

CENTRO SOFTWARE

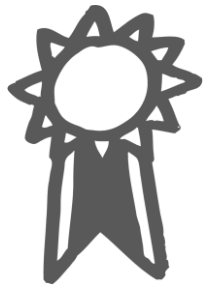
The ERP Company

L’azienda

CENTRO SOFTWARE

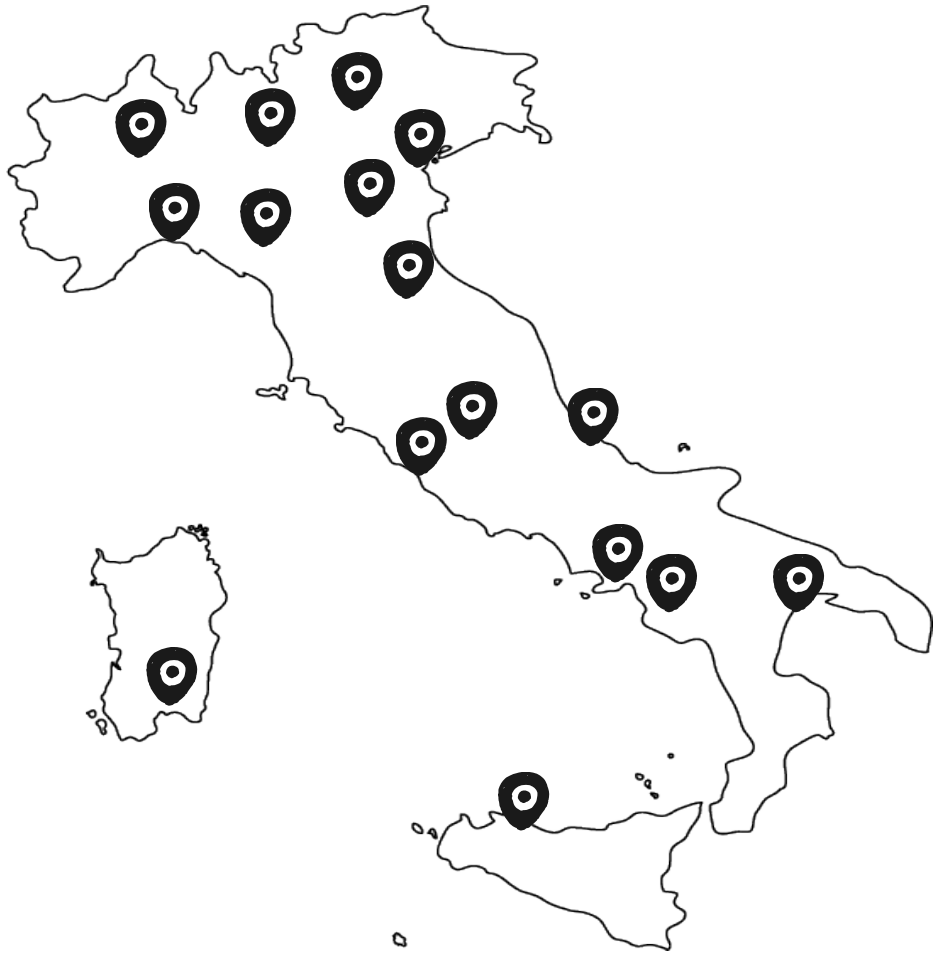
The ERP Company

Centro Software, fondata nel 1988, Progetta e Produce il miglior sistema **ERP di 2° generazione** per le imprese che **producono** e **esportano**!



L’azienda

Presenti in Italia con oltre **200 tecnici e consulenti** ed oltre **53 business partner** sparsi su tutto il territorio



Gli ERP di Centro Software

Sono utilizzati da oltre **2700 clienti** con filiali operative **in tutto il mondo**

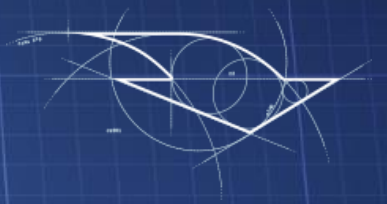
Ogni giorno in Italia più di **31.000** persone lavorano e producono con i nostri software

Oltre **16,5 miliardi** di Euro gestiti complessivamente in Italia

Perché introdurre in un’azienda SAM ERP2 ?

Per poter rispondere a queste domande

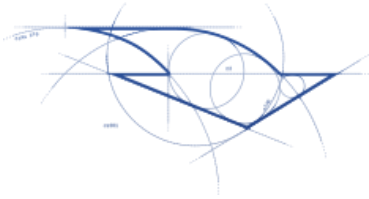
- Che cosa produrre?
- Quali risorse occorrono per farlo?
- Possiamo confermare al cliente la data richiesta di consegna?
- Se accettiamo anche quest’ordine cosa va in ritardo?
- E’ possibile anticipare quest’ordine in N giorni?
- Su quale fatturato potrò contare questo mese?
- Abbiamo la copertura finanziaria necessaria?
- Qual è il «quarto» margine della commessa?



Lo sviluppo di applicazioni Enterprise

Cosa si intende per Applicazione Enterprise?

- Il termine Enterprise non è riferito alle dimensioni dell'azienda a cui è destinato il software
  - Notepad.exe scritto per Fiat Chrysler sarebbe un'applicazione Enterprise ☺
- Martin Fowler: "Enterprise applications are about the display, manipulation, and storage of large amounts of often complex data and the support or automation of business processes with that data"



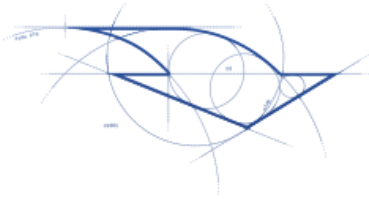
Lo sviluppo di applicazioni Enterprise..

- Necessita di una strategia!

- Perché?
  - Più sviluppatori lavorano simultaneamente
  - Quando un’oggetto, una classe, una funzione, una procedura.. fa troppe cose, il problema non è troppo lontano!
  - L’incubo inizia ancor prima del rilascio

Ricorda…

- Il codice già scritto non è sbagliato by default
- E’ stato influenzato
  - Dalle strategie aziendali!
  - Dallo stato d’animo di chi lo ha scritto
  - Dalla tecnologia utilizzata e da quanto era conosciuta

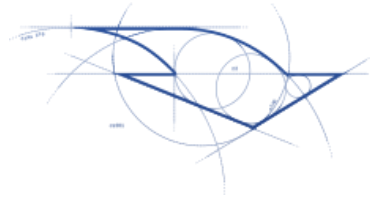


I sintomi di un’architettura sbagliata

- Ci sono quattro sintomi primari che identificano un’architettura *sbagliata*
  - Rigidità
  - Fragilità
  - Immobilità
  - Viscosità

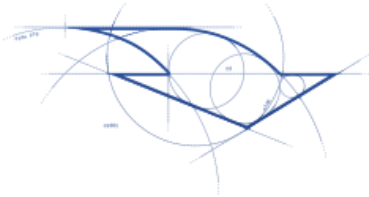
Rigidità

- Le modifiche al software sono difficili da attuare
- Una modifica, stimata in qualche giorno di lavoro, diventa una maratona di modifiche, poi le modifiche non sono stabili
- Tempi di sviluppo non noti



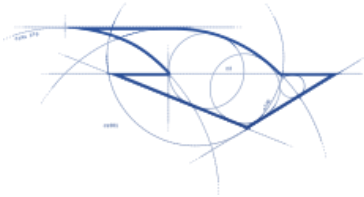
Fragilità

- Rappresenta la tendenza ai malfunzionamenti ogni volta che il software viene modificato
- I malfunzionamento avvengono in aree, che concettualmente, non hanno relazioni con l'area modificata
- Il software diventa impossibile da manutenere
- Le fix aggiungono più problemi di quanti ne risolvono



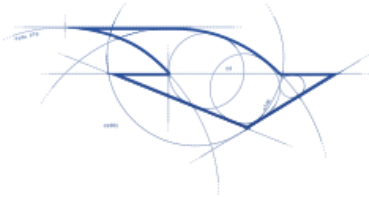
Immobilità

- Impossibilità di riutilizzare parti del software all'interno di altri progetti
- Il numero di dipendenze è talmente elevato da impedire l'estrazione di un modulo
  - Quindi il codice viene duplicato (!!)



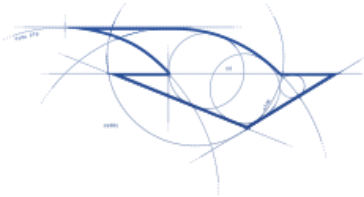
Viscosità

- Si manifesta in due forme
  - Viscosità di design
  - Viscosità di ambiente
- Viscosità di design
  - Costringe ad utilizzare la strada più veloce ad ogni blocco
- Viscosità di ambiente
  - Costretti a ricompilare sempre tutto, lentezza nelle procedure



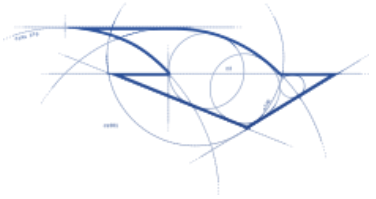
Gli ingredienti

- I principi SOLID
- Object Oriented Programming (OOP)
- Utilizzo dei Design Pattern
- Source control
- Naming convention
- Unit testing



I principi SOLID

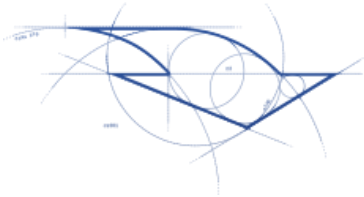
- S = Single responsibility principle (SRP)
- O = Open/close principle (OCP)
- L = Liskov substitution principle (LSP)
- I = Interface segregation principle (ISP)
- D = Dependency injection principle (DIP)



La strategia

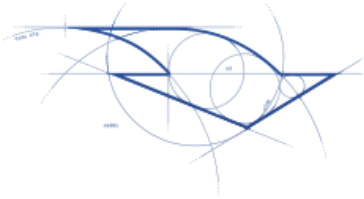
- Separare le responsabilità

- Esistono diversi modelli (= Design Pattern) che ci aiutano a separare le responsabilità e permettono di risolvere problemi ricorrenti



Design Pattern

- Cosa sono
- Da dove provengono, quali sono le origini
- Perché sono importanti

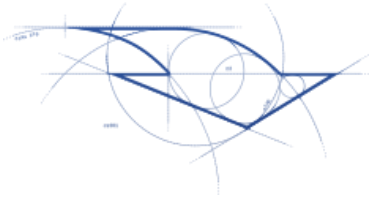


Cosa sono i Design Pattern?

- Soluzioni generali e riutilizzabili per risolvere problemi ricorrenti nello sviluppo software
- Un nome specifico li identifica univocamente
- Sono Half Cooked

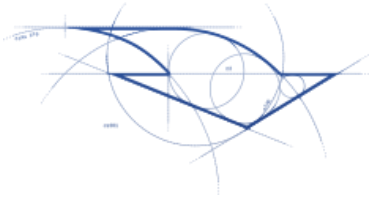
- Non sono un framework, o una DLL
- Non sono soluzioni definitive

I Design Pattern si occupano di..



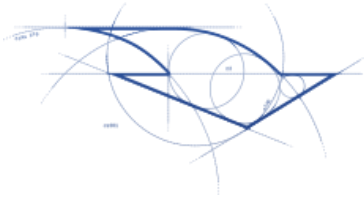
- Architettura del software
- Astrazioni sul codice
- Relazioni tra le classi o altri oggetti
- Soluzioni a problemi ricorrenti

I Design Pattern NON occupano di..



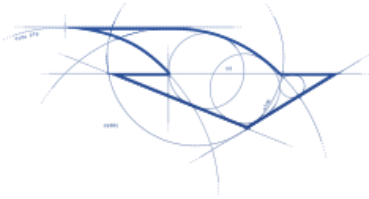
- Algoritmi
- Implementazioni specifiche
- Classi specifiche
- Spiegare ogni singola riga di codice

- Il Quicksort è un algoritmo, non è un Design Pattern!



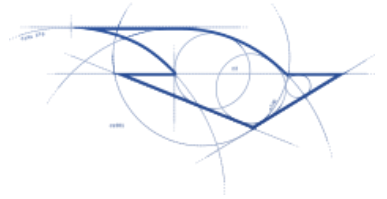
Design Pattern: Perché sono importanti?

- Evitano di re-inventare costantemente soluzioni a problemi noti
- Forniscono il punto di partenza per la soluzione
- Aumentano la produttività in team
- Agevolano il riutilizzo del codice
- Permettono di applicare Unit Testing



Design Pattern: Il costo

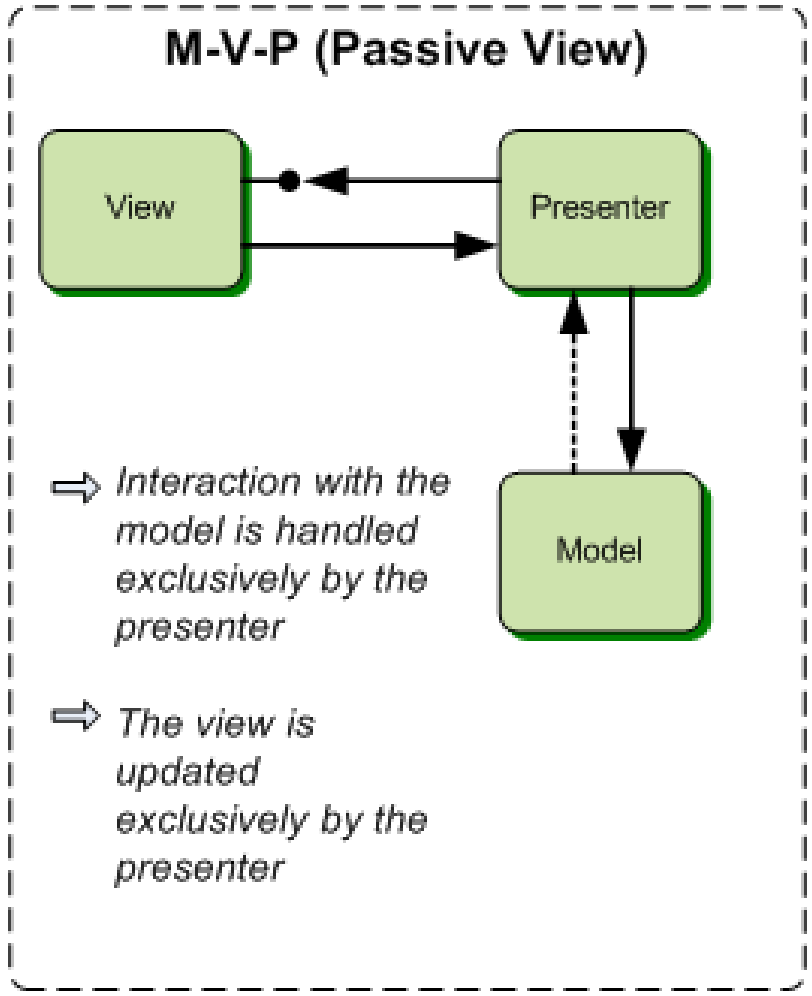
- Lunga curva di apprendimento
- Senza una chiara comprensione di quello che ogni entità dovrebbe fare, ma soprattutto **non fare**, il codice dovrà essere modificato più volte per correggere gli errori
- Quello che oggi facciamo velocemente, in una sola classe, domani richiederà più classi, più specifiche
  - Può sembrare uno svantaggio, ma non è così!



M-V-P Design Pattern: Vantaggi

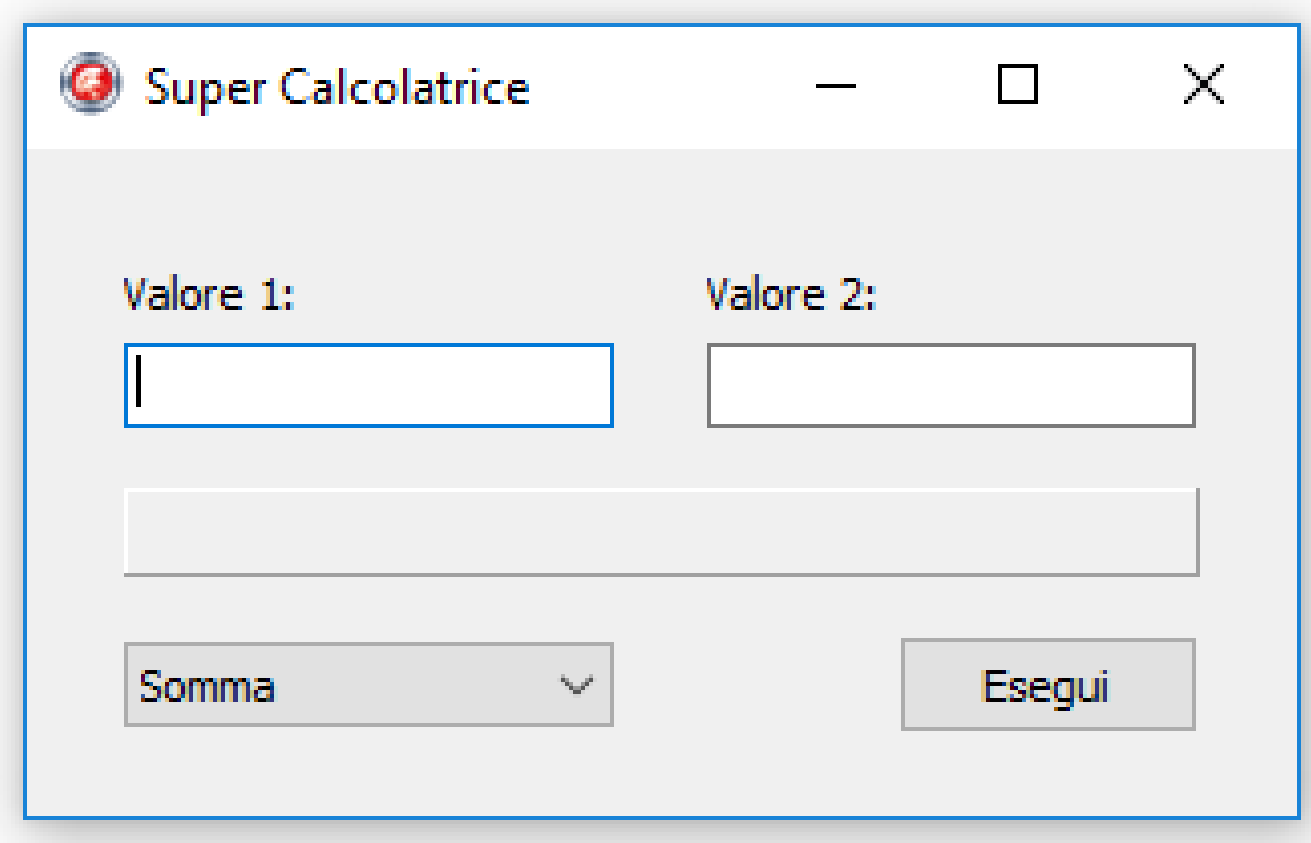
- **Disaccoppia** la logica di business (BL) dalla logica di presentazione (PL)
- Permette il **riutilizzo** del codice
- Consente l'esecuzione di **test** automatici (= Unit Test)

M-V-P Design Pattern (Passive View)



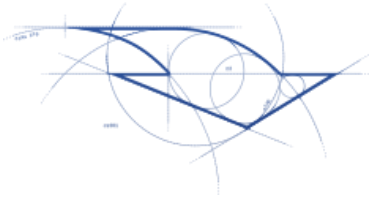
- **View**: Ha la responsabilità di presentare i dati (PL)
- **Presenter**: Comunica con la view attraverso la sua interfaccia e implementa la Logica di Business (BL)
- **Model**: Ha la responsabilità di gestire la memorizzazione dei dati

Implementiamo una calcolatrice



Implementiamo una calcolatrice

```
procedure TFCalculator.btnEseguiClick(Sender: TObject);
begin
  if (ComboBox.Text = '+') then
    edtResult.Text :=
      IntToStr(StrToInt(edtValore1.Text) +
               StrToInt(edtValore2.Text))
  else
    edtResult.Text :=
      IntToStr(StrToInt(edtValore1.Text) -
               StrToInt(edtValore2.Text));
end;
```



DEMO

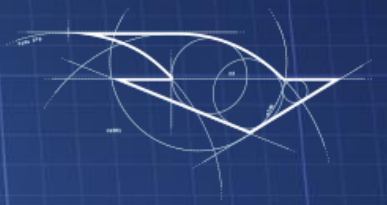
Realizziamo l’applicazione Calcolatrice applicando il Design Pattern M-V-P

% di variazione del codice

Calcolato su una unit..

$$I = \frac{Numero\ di\ condizioni\ nella\ PL}{Numero\ di\ righe\ totali} * 100$$

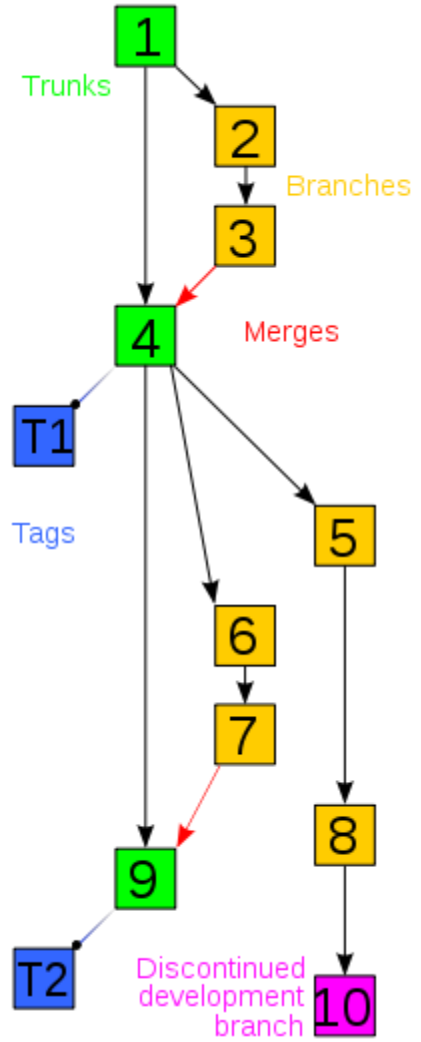
Nella Presentation Logic (form, view, ecc..) la percentuale deve tendere a zero!



Source control

Source control

- Il controllo versione (versioning), in informatica, è la gestione di versioni multiple di un insieme di informazioni.
- Gli strumenti software per il controllo versione sono ritenuti molto spesso necessari per la maggior parte dei progetti di sviluppo software



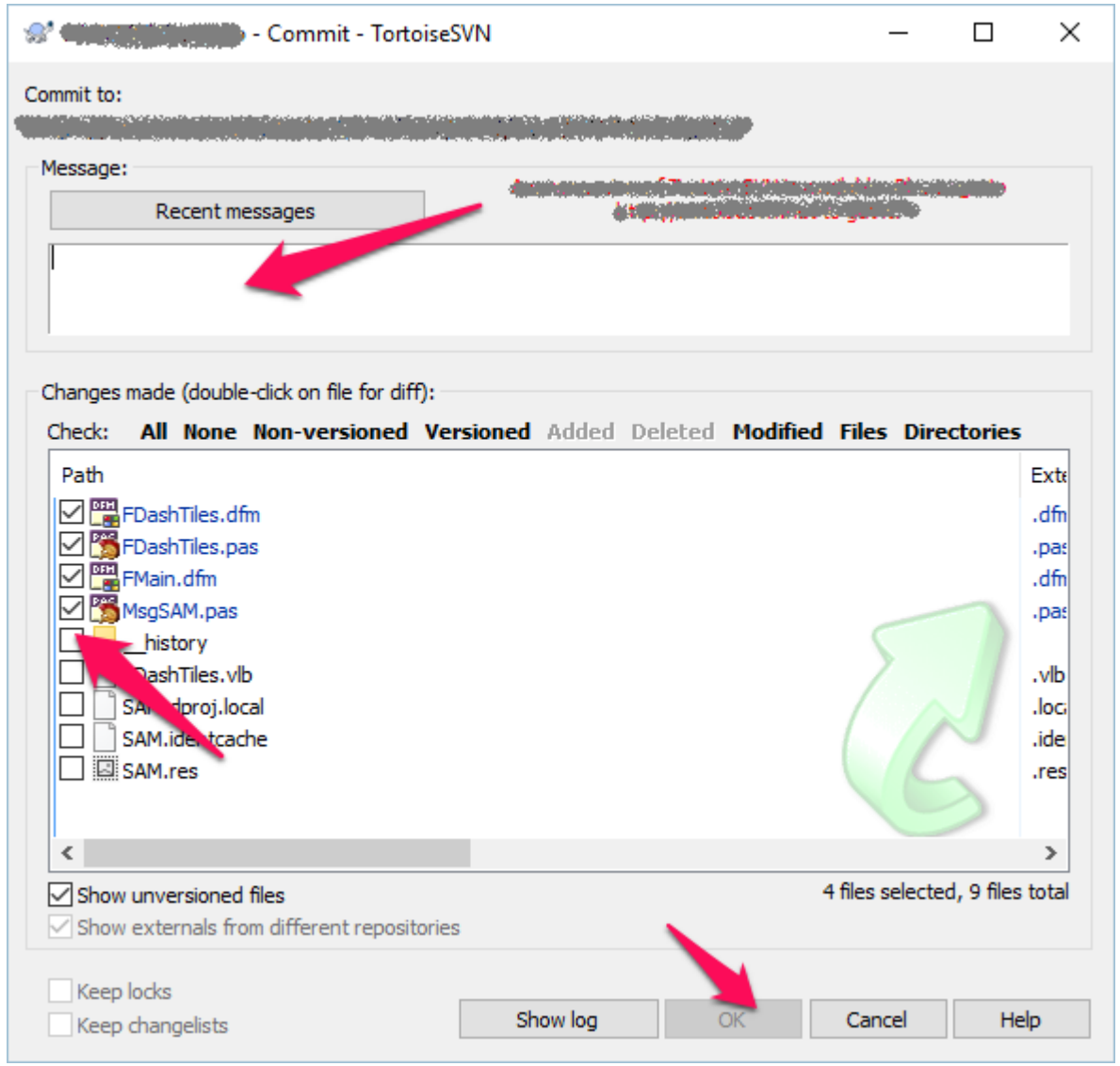
https://it.wikipedia.org/wiki/Controllo_versione

Procedura di commit

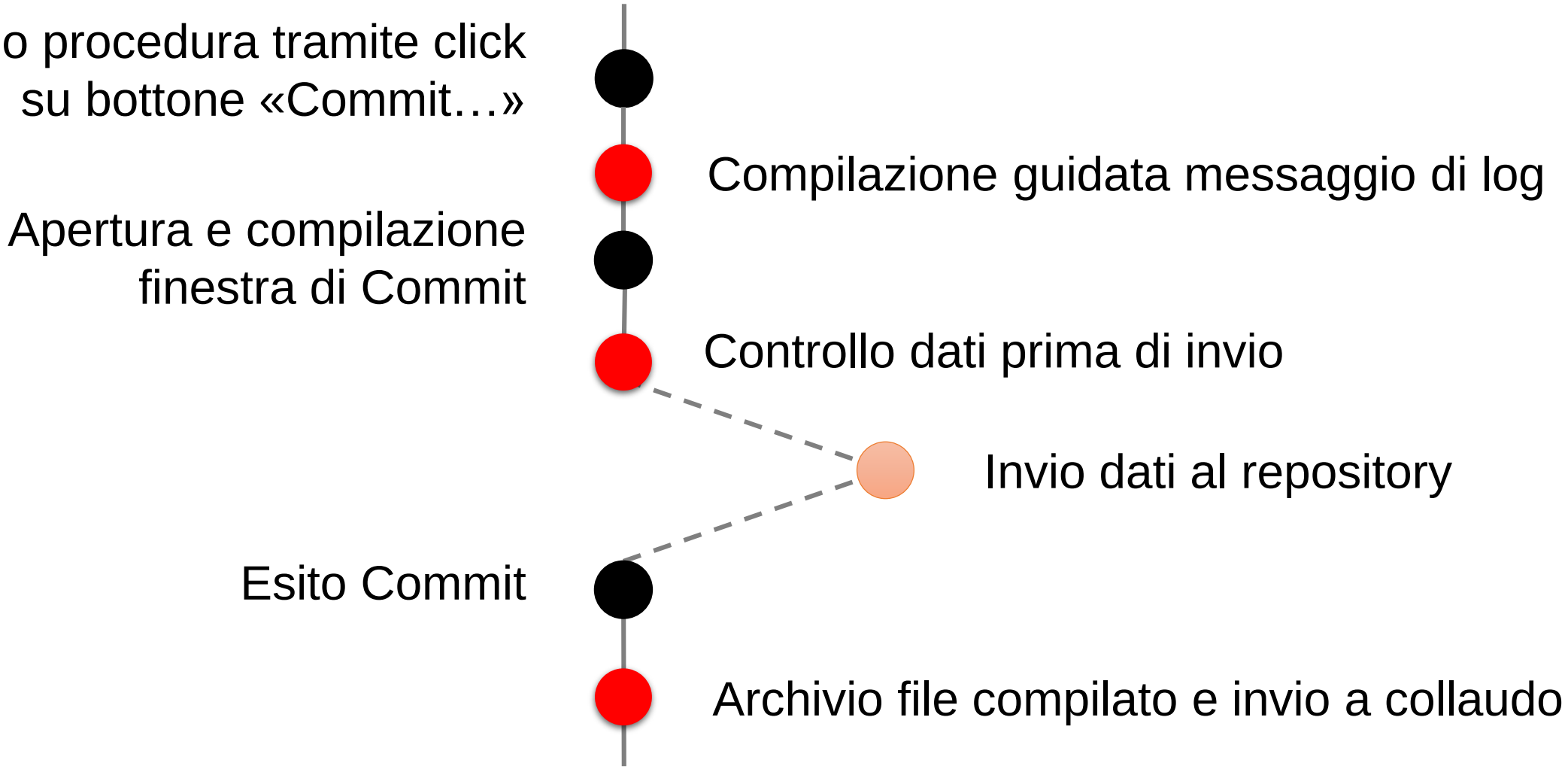
Inizio procedura tramite click su bottone «Commit…»

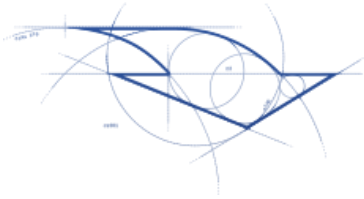
Apertura e compilazione finestra di Commit

Esito Commit



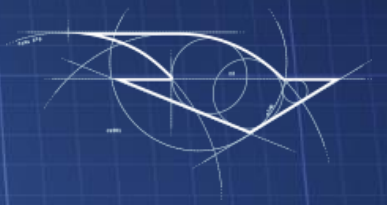
Procedura di commit





Template per i commit

- Summary
  - Verb
  - Fragment
  - Ticket #
- Details
  - Why
  - Module information
  - Dependency



Naming conventions

Naming conventions: Definizione

"In computer programming, a naming convention is a **set of rules** for choosing the character sequence to be used for identifiers which denote **variables**, **types**, **functions**, and **other entities** in source code and documentation"

Naming convention (programming)

Naming conventions

- Tutti i parametri in ingresso a un metodo devono iniziare con la lettera «A»

```
procedure TMarca.Copia(ObjToCopy:
TMarca);
begin
   // Copia dei dati
   Campi := ObjToCopy.Campi;
   // copia delle proprietà
   FoundP := ObjToCopy.FoundP;
end;
```

Naming conventions

- Tutti i parametri in ingresso a un metodo devono iniziare con la lettera «A»

```
procedure TMarca.Copia(AObjToCopy:
TMarca);
begin
   // Copia dei dati
   Campi := AObjToCopy.Campi;
   // copia delle proprietà
   FoundP := AObjToCopy.FoundP;
end;
```

Naming conventions

- Tutte le variabili locali a un metodo devono iniziare con la lettera «L»

```
procedure TMarca.Incolla;
var OldID, idxObj: Integer;
begin
  idxObj := ArrayCopia.IndexOf('ObjMarche');
  if idxObj <> -1 then
  begin
    OldID := Campi.ID;
    // Incolla l'oggetto di ArrayCopia
    //…
  end;
end;
```

Naming conventions

- Tutte le variabili locali a un metodo devono iniziare con la lettera «L»

```
procedure TMarca.Incolla;
var LOldID, LidxObj: Integer;
begin
  LidxObj := ArrayCopia.IndexOf('ObjMarche');
  if LidxObj <> -1 then
  begin
    LOldID := Campi.ID;
    // Incolla l'oggetto di ArrayCopia
    //…
  end;
end;
```

Naming conventions

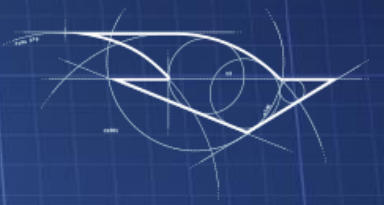
- Tutte le variabili locali a una classe devono iniziare con la lettera «F»

```
procedure TMarca.Incolla;
var LOldID, LidxObj :Integer;
begin
  idxObj := ArrayCopia.IndexOf('ObjMarche');
  if LidxObj <> -1 then
  begin
    LOldID := Campi.ID;
    // Incolla l'oggetto di ArrayCopia
    //…
  end;
end;
```

Naming conventions

- Tutte le variabili locali a una classe devono iniziare con la lettera «F»

```
procedure TMarca.Incolla;
var LOldID, LidxObj :Integer;
begin
  idxObj := ArrayCopia.IndexOf('ObjMarche');
  if LidxObj <> -1 then
  begin
    LOldID := FCampi.ID;
    // Incolla l'oggetto di ArrayCopia
    //…
  end;
end;
```

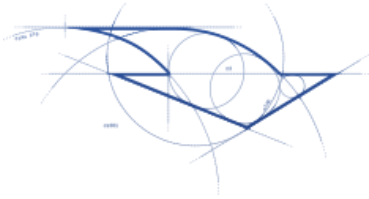


Set based thinking

Il linguaggio SQL, T-SQL per SQL Server

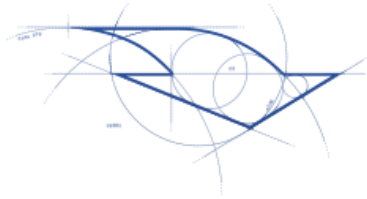
Il linguaggio T-SQL

- Deriva dal linguaggio SQL ANSI
- Confrontato con gli altri linguaggi
  - Non è difficile da imparare
  - Può essere molto tollerante
- Chi sviluppa, lo utilizza ogni giorno per
  - SELECT, INSERT, UPDATE, DELETE, CREATE, ecc..
- Solo chi investe più tempo scopre la sua natura dichiarativa!



Set-Based or Iterative Code?

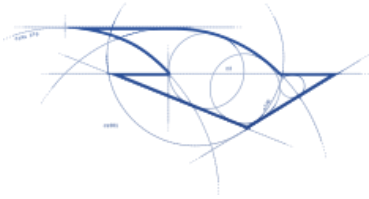
- L'approccio set-based non è intuitivo
- Per realizzare soluzioni set-based è necessario conoscere i fondamenti del linguaggio SQL
- Le difficoltà che si incontrano nel pensare set-based derivano spesso dal nostro background



Set-Based or Iterative Code?

Quante volte abbiamo lavorato con un file?

```
open file
fetch first
while not EOF
begin
  process
  fetch next
end
```



Set-Based or Iterative Code?

- Siamo abituati a pensare che
  - I dati siano memorizzati in un ordine specifico
  - Le operazioni di fetch
    - Garantiscono la restituzione ordinata di porzioni di dati
    - Una porzione alla volta
- La nostra mente è programmata per pensare ai dati in questi termini
  - In modo ordinato
  - Manipolati a porzioni, una porzione alla volta

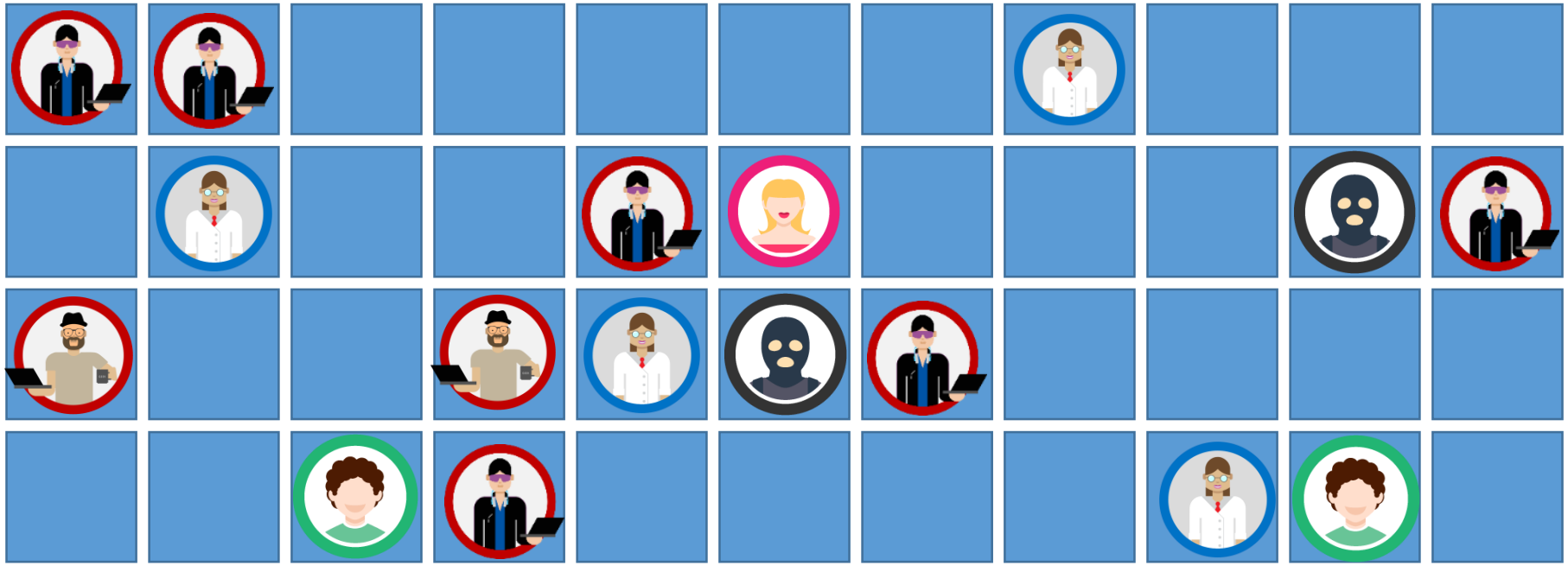
Set-Based or Iterative Code?

- Sono due approcci per sviluppare soluzioni
  - Conosciuti entrambi da chi sviluppa
  - Il più utilizzato, però, è l'approccio procedurale ☹
- Perché l'approccio set-based è consigliato?
- Perché molti sviluppatori utilizzando l'approccio procedurale?
- Quali sono gli ostacoli che impediscono l'utilizzo dell'approccio set-based?

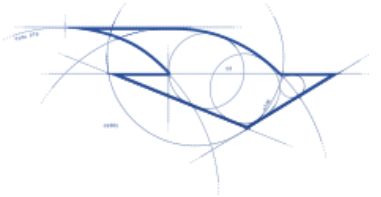
Esercizio

- Dato un gruppo di persone il cui numero è definito, trovare in quale fila di un cinema, o di un teatro, vi è sufficiente spazio affinché queste persone possano avere tutti i posti a sedere vicini

Esercizio: Posti Liberi



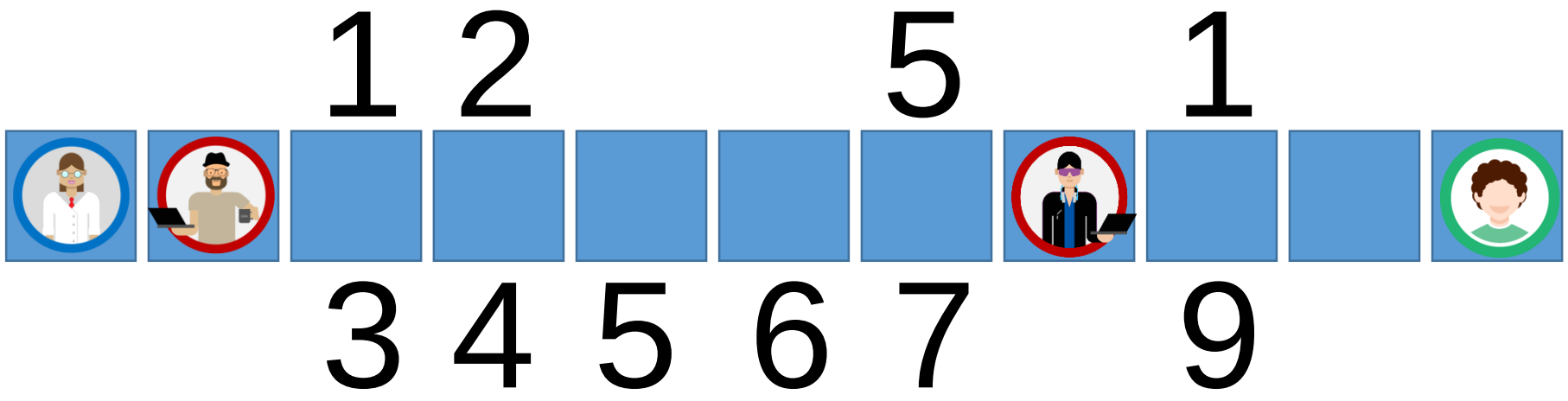
Cerchiamo di risolvere il problema..



Posti Liberi: Prima soluzione

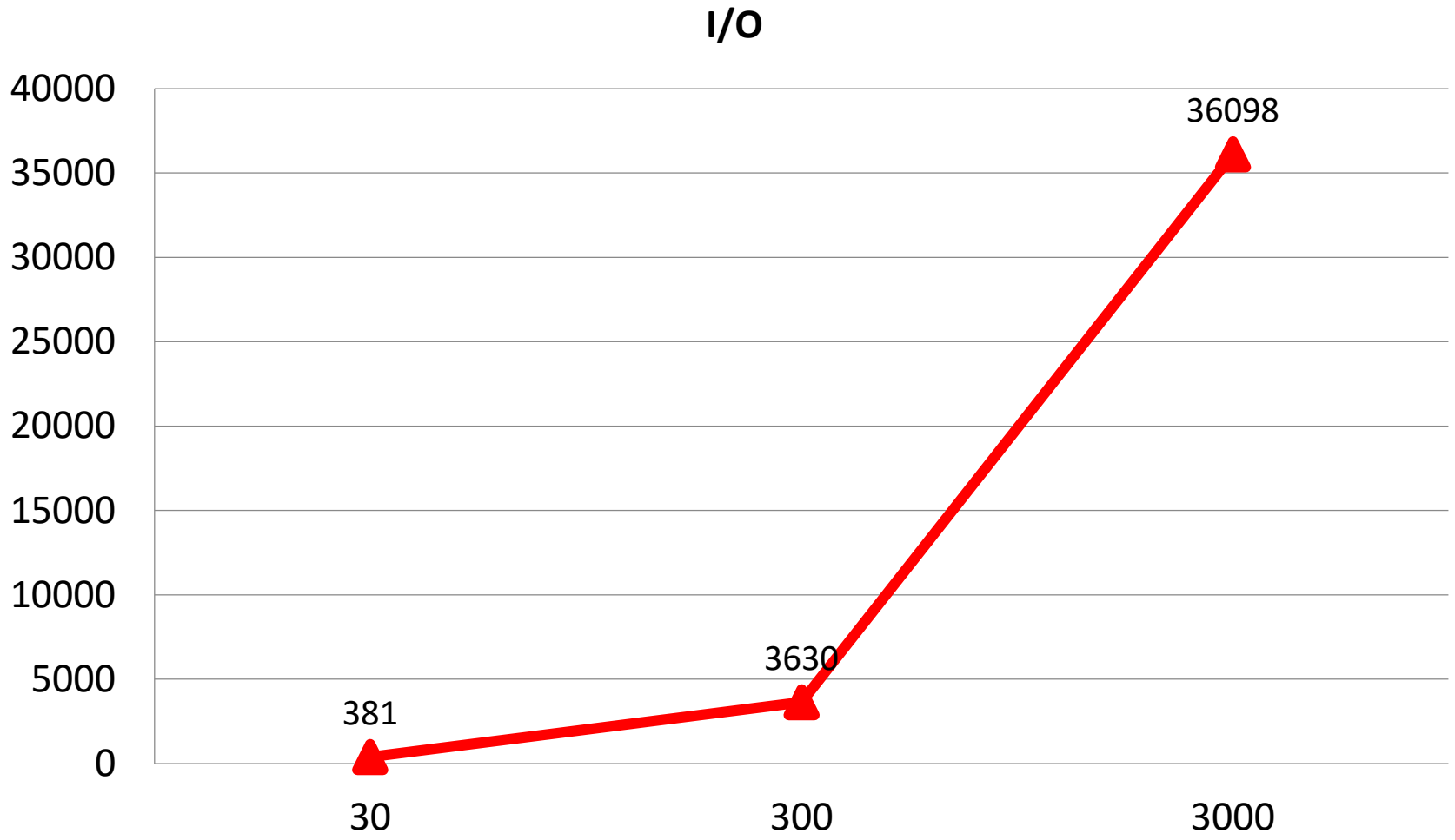
Misuriamo le performance e diamo un’occhiata al codice

Posti Liberi – Prima Soluzione

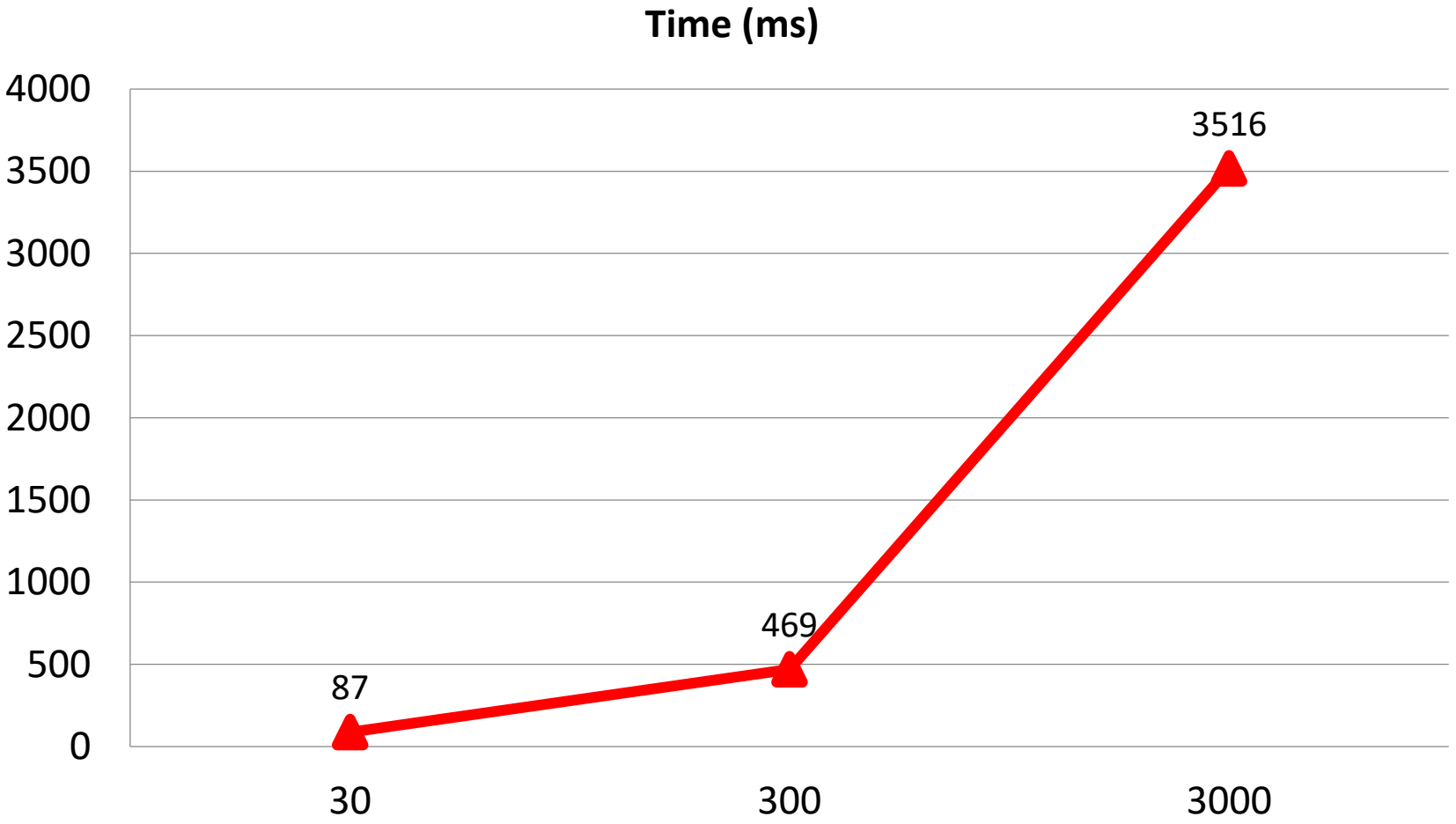


A. Vai al primo posto libero
B. Imposta il contatore posti liberi (NPostiLiberi) a 1
C. Parti dal primo posto libero (n), memorizza questo numero come primo spazio libero (NPrimoLibero)
D. Se il prossimo posto libero è (n+1) allora aumento NPostiLiberi
E. Se il prossimo posto libero non è (n+1) allora conservalo ancora con NPrimoLibero
F. Riparti dal punto A

Prestazioni I/O

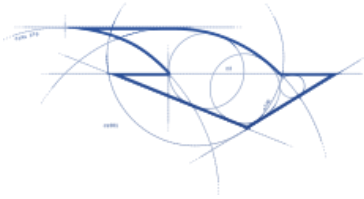


Prestazioni Time



La prima soluzione

- Per ogni riga
- E’ **davvero** la soluzione migliore?
- E’ la soluzione più semplice?
- Non avete la sensazione che ci potrebbe essere qualcosa di meglio? Un’altra soluzione per il problema?



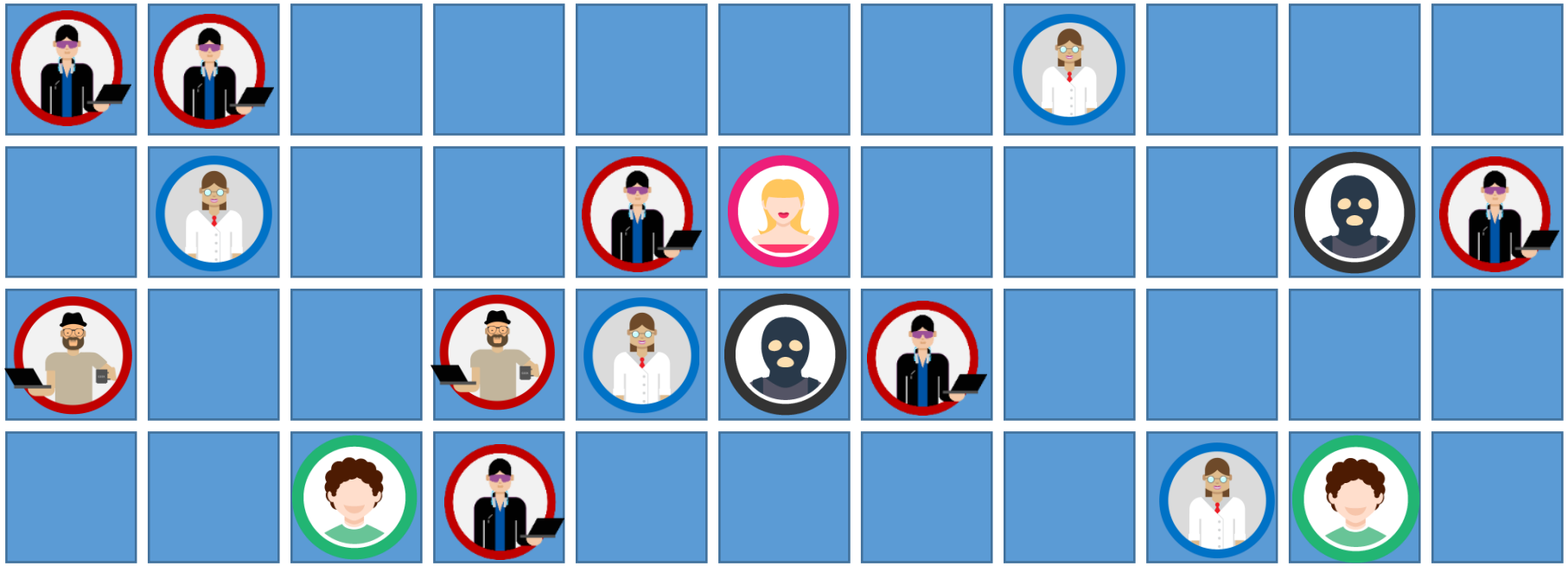
La prima soluzione

- Il risultato non è così buono ☹
- Non è poi così facile ☹
- Cerchiamo di pensare al di fuori del codice
- Il primo passo è: fermarsi di pensare al codice!
  - Il codice è solo uno «strumento»
- Proviamo a trovare la soluzione dal punto di vista puramente logico

Pensiamo Set-based

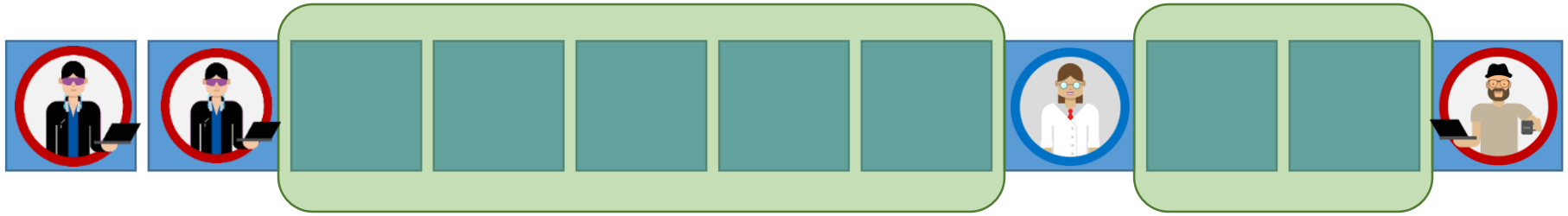
- Solitamente iniziamo immediatamente a realizzare la prima soluzione che ci viene in mente
  - ..e poi risolviamo tutti i problemi che ne derivano
  - Non sarà in grado di scalare
  - Non offrirà buone prestazioni
- Abbandoniamo questo approccio e cerchiamo di trovare il miglior algoritmo (indipendente dal linguaggio) che sia in grado di risolvere il problema

Esercizio: Posti Liberi



Posti Liberi: Seconda soluzione

Posti Liberi – Set-based



- Se l'RDBMS (SQL Server nel nostro caso) avesse gli occhi ☺ potrebbe vedere l'immagine qui sopra e sarebbe in grado di osservare che ci sono **gruppi** di spazi liberi
- Abbiamo bisogno di trasformare questa immagine "solo per chi può vederla" in qualcosa che possa essere gestito dal DB

Posti Liberi – Set-based

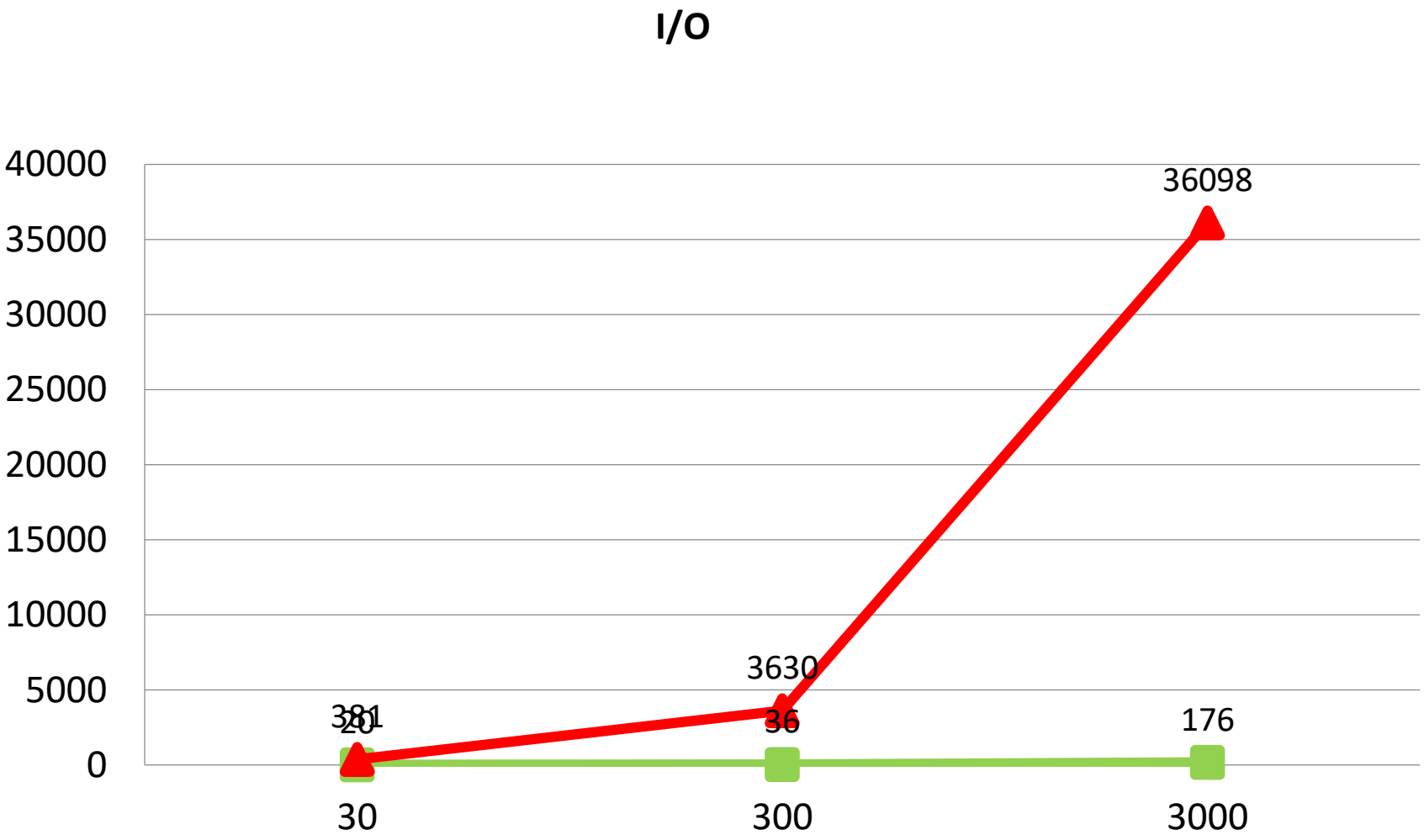
1 2 3 4 5 6 7 8 9 10 ...

1 2 3 4 5 6 7 ...

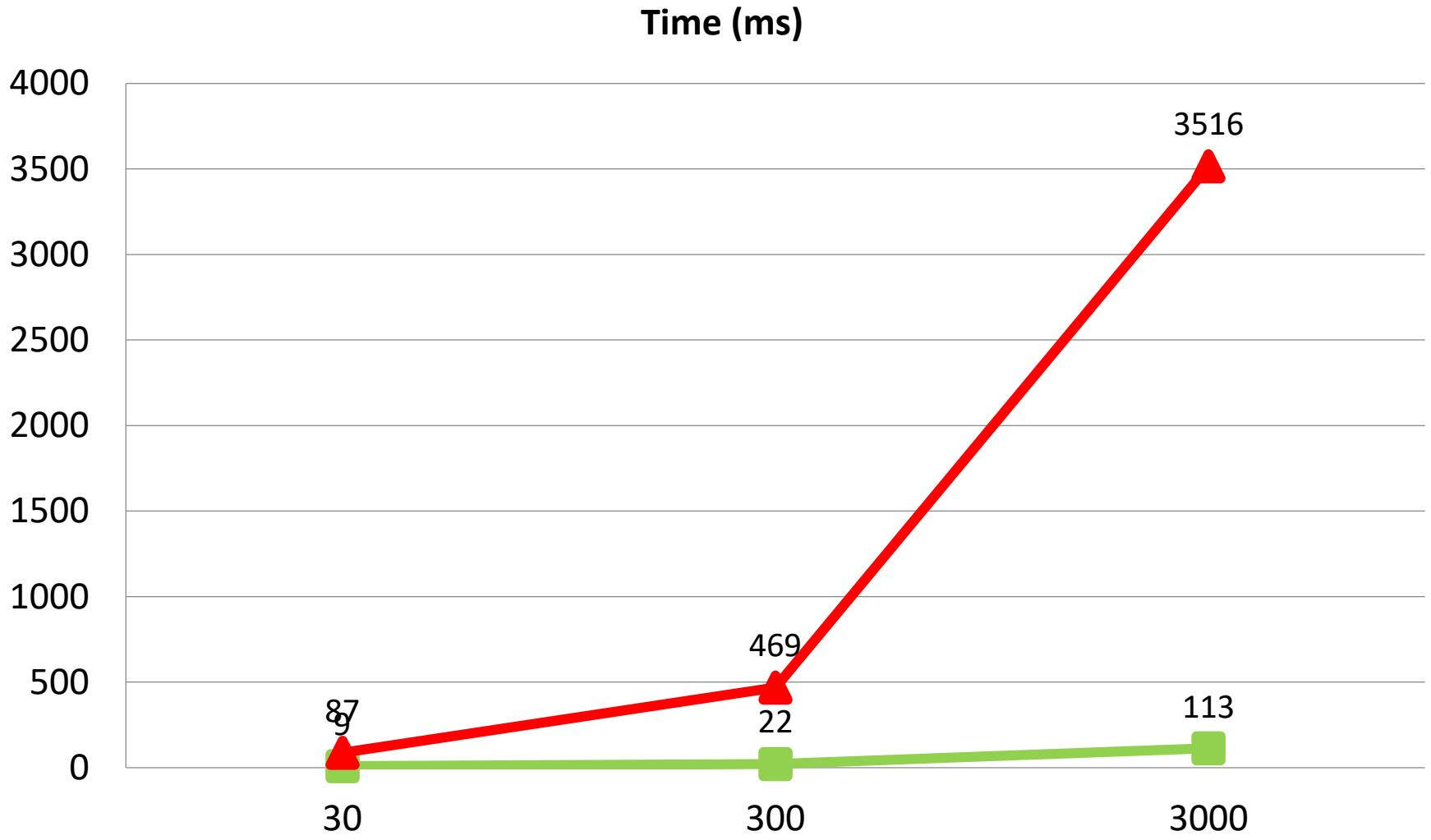
2 2 2 2 2 3 3

A. Enumeriamo tutti i posti (liberi e occupati)
B. Enumeriamo tutti i posti liberi
C. Facciamo una semplice sottrazione..
D. ...ora possiamo utilizzare la clausola GROUP BY ☺

Confronto Prestazioni



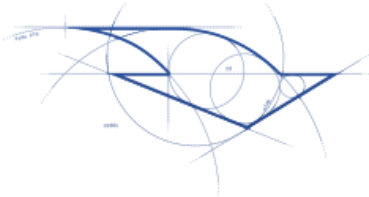
Confronto Prestazioni



We hire!

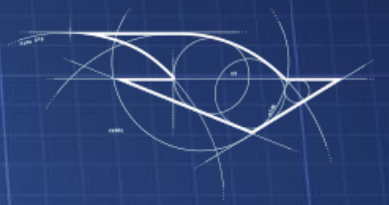
Ricerchiamo, per la sede di San Pietro in Casale (BO), Sviluppatori Software

https://www.linkedin.com/jobs/view/513704113/



Summary

- Lo sviluppo di applicazioni Enterprise necessita di una strategia!
- Gli ingredienti
  - Principi SOLID
  - OOP
  - Design Pattern
  - Source control
  - Naming convention
  - Unit testing
  - Set-based thinking



Grazie!