INTRODUCTION TO DESIGN PATTERNS

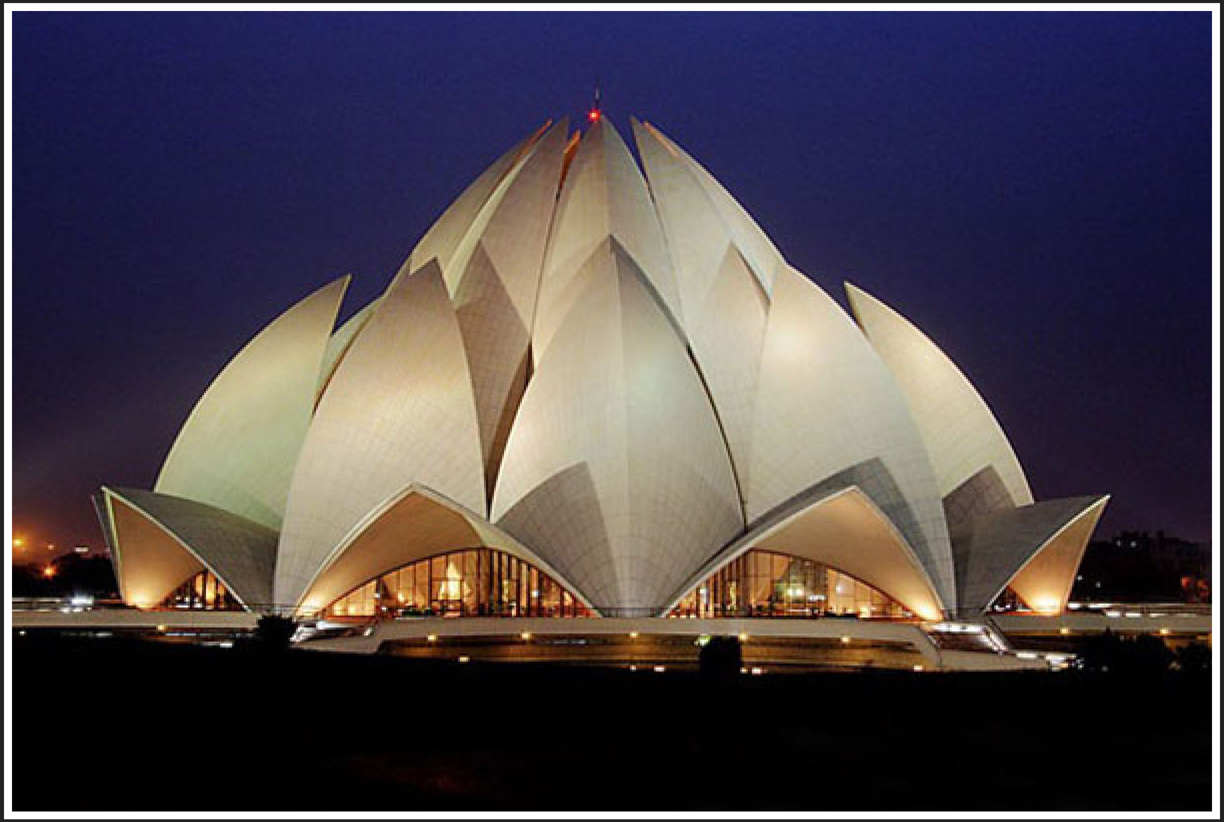
DR. VIVEK NALLUR

VIVEK.NALLUR@SCSS.TCD.IE

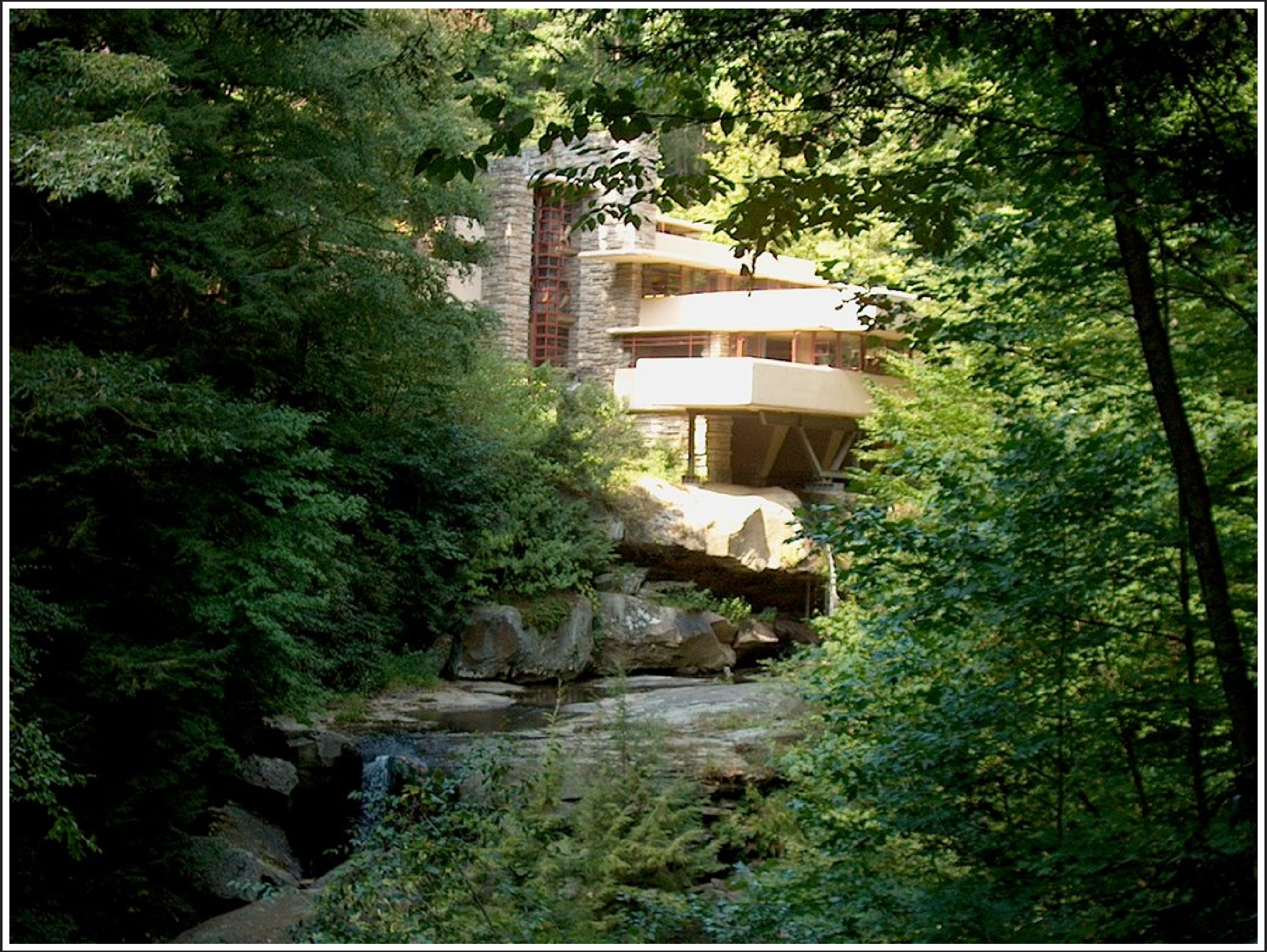
OUTLINE OF THIS TALK

- What are Design Patterns?

BEAUTIFUL STRUCTURES

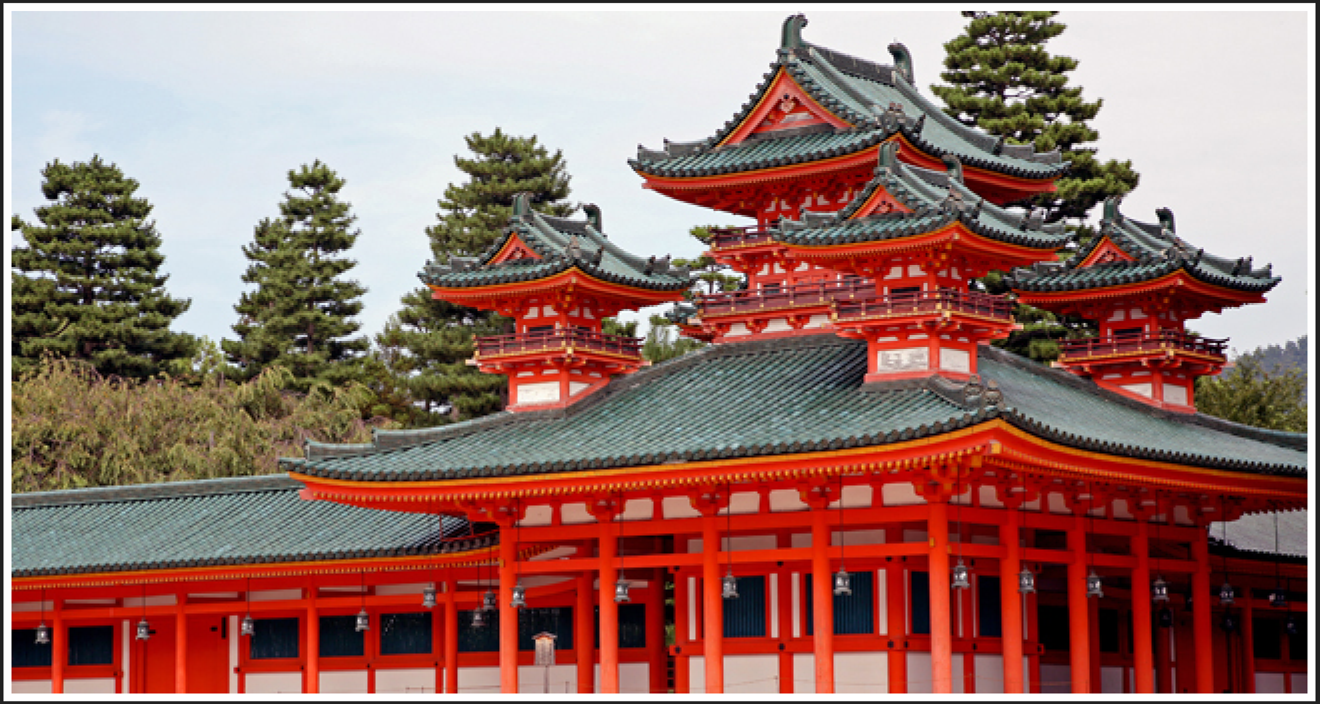


BEAUTIFUL STRUCTURES - II



BEAUTIFUL STRUCTURES - III

WHY ARE THESE BEAUTIFUL?

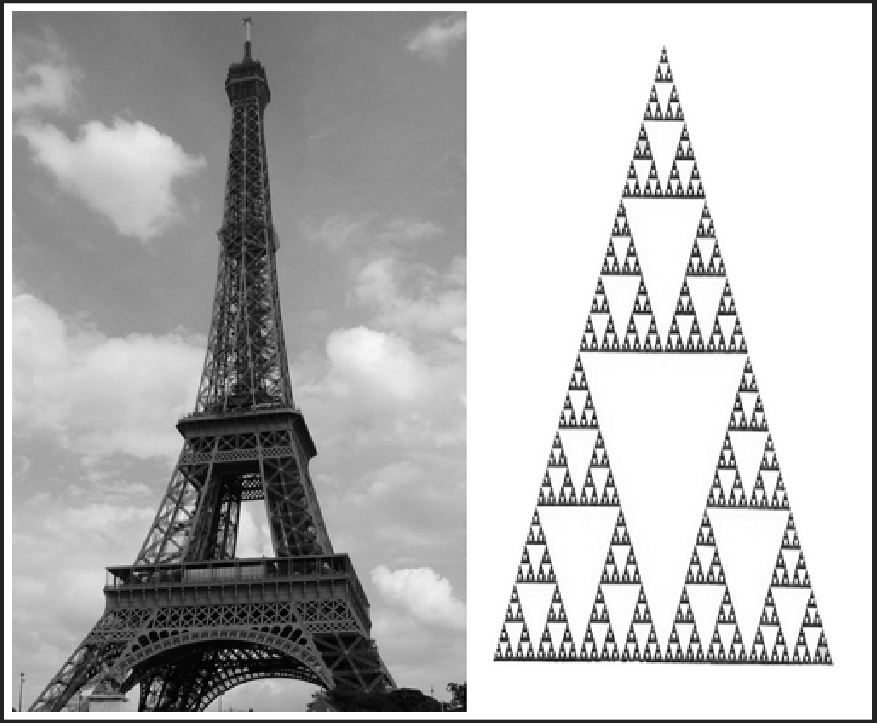


SOME ELEMENTS OF BEAUTY

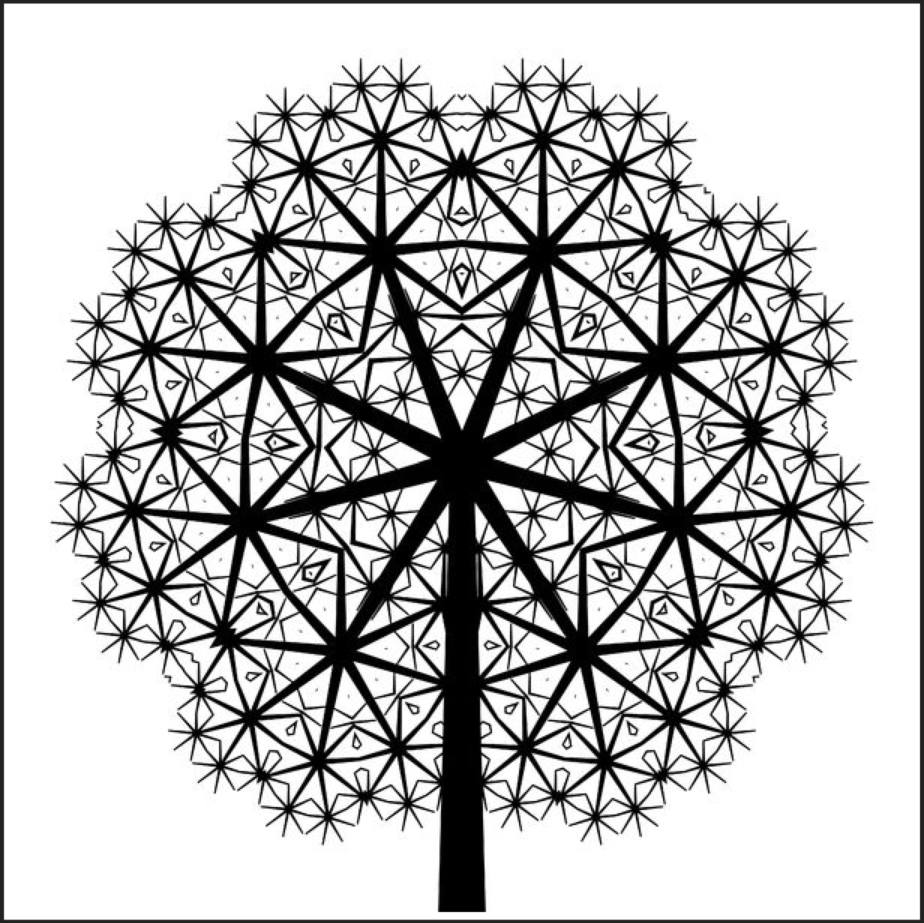
- Harmony
- Simplicity
- Function and form are inter-linked

HOW TO CREATE A PLEASING STRUCTURE?

- Identify few simple patterns
- Ensure all forces are balanced
- Rinse, repeat



The fractal nature of life



BUILDING SOFTWARE

- Is quite a bit like creating a building
- Many forces to take into account
- Many variables that change over time

NOTION OF DESIGN PATTERNS

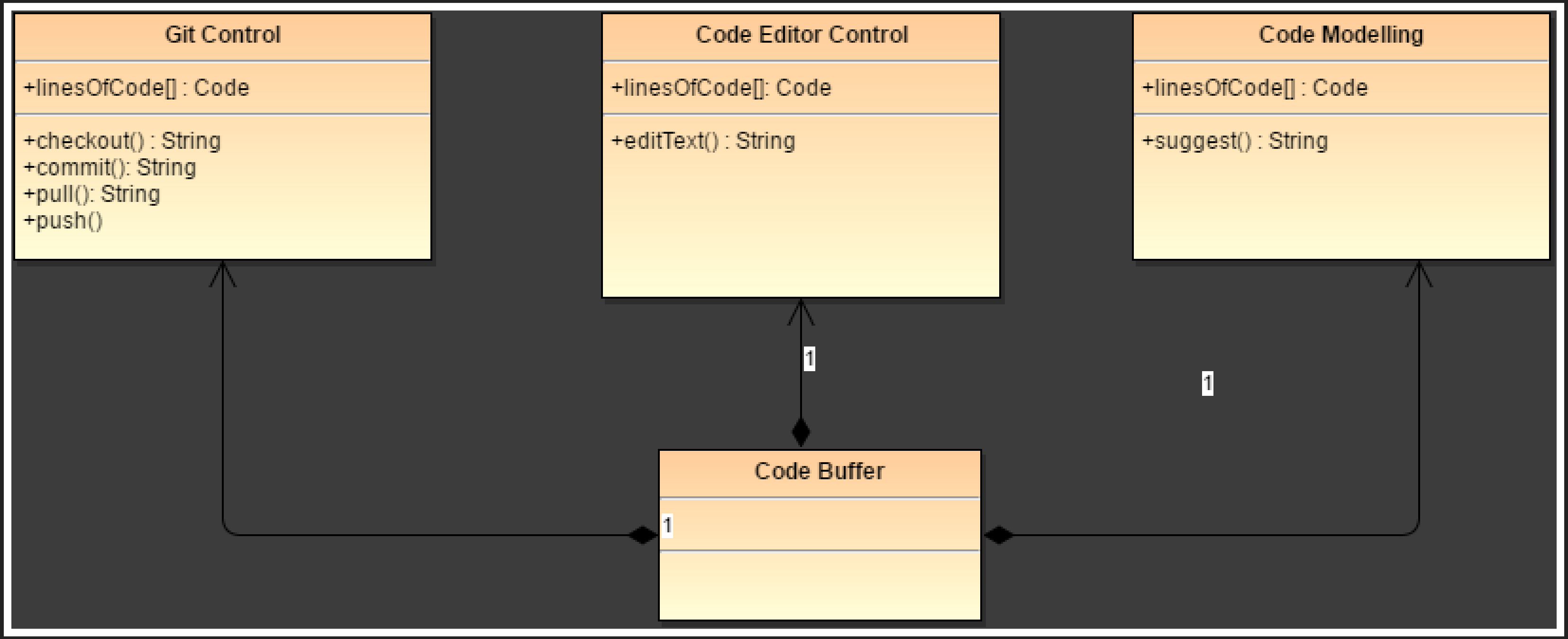
- Evolved from 'real building terminology'
- Allows certain elegant solutions, in the presence of:
  - Common problems
  - Common tools
  - Common forces of change

LET'S START WITH A USECASE

- Create a custom IDE for an enterprise that develops android apps. The IDE must:
  - Identify syntax errors
  - Intelli-suggest methods
  - Commit code to a common repository
  - Deploy built code onto device

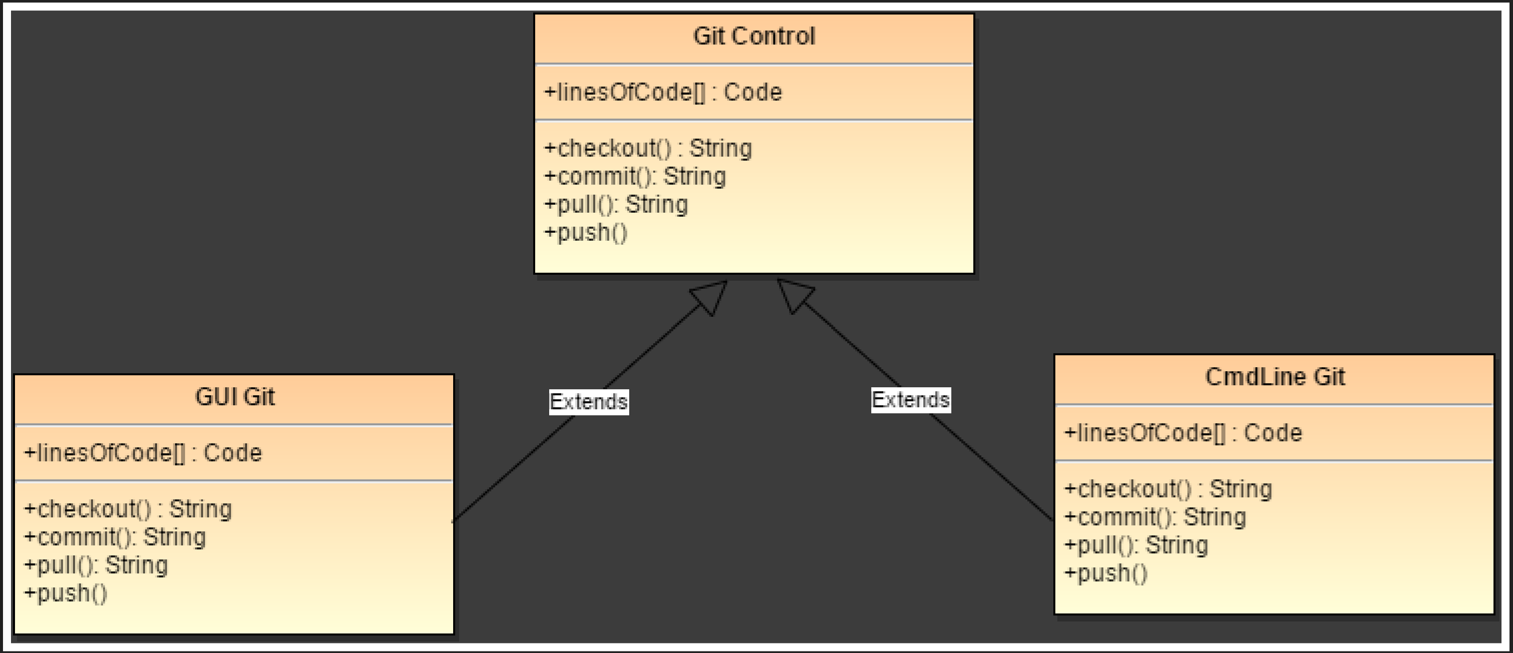
A PIECE OF CAKE

<u>C</u>ommon <u>A</u>pp development <u>K</u>it for the <u>E</u>nterprise



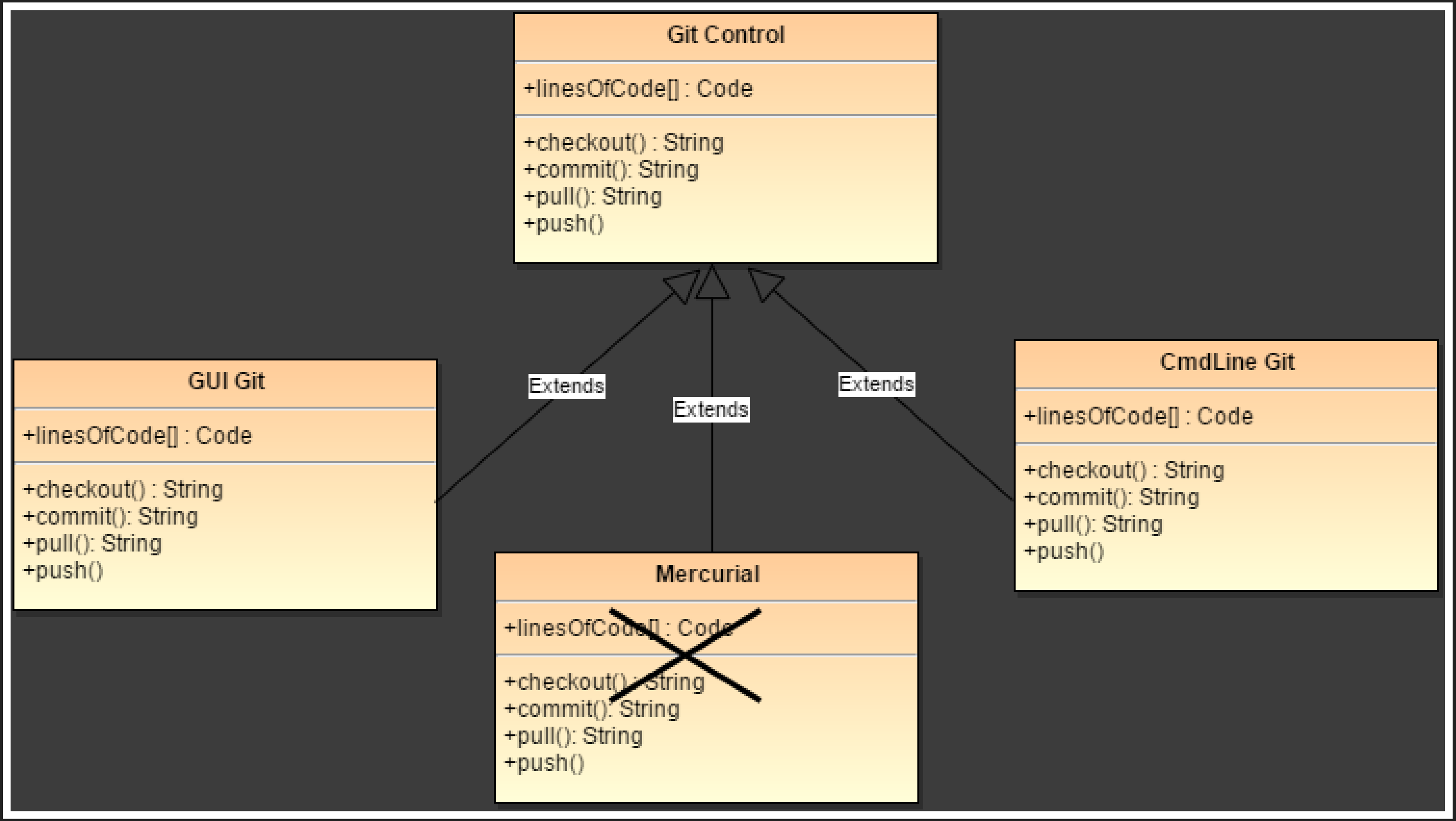
CLIENT WANTS MULTIPLE 'CLIENTS'

Easily solved using *inheritance*



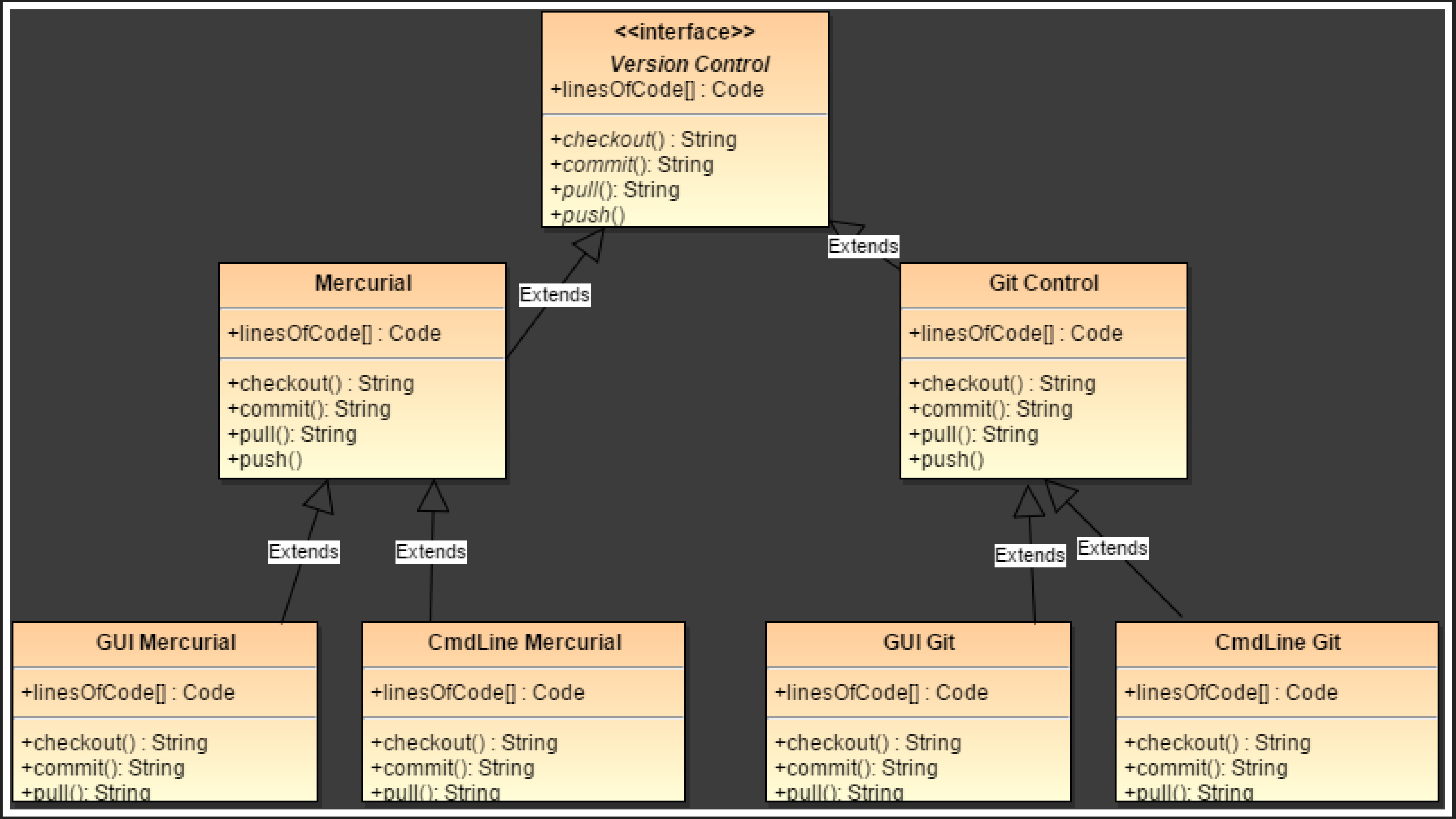
CLIENT WANTS MULTIPLE REPOSITORIES

Some developers use *Mercurial* instead of *Git*



SUPERTYPES, ABSTRACT CLASSES, ETC.

Every self-respecting programmer programs to an interface



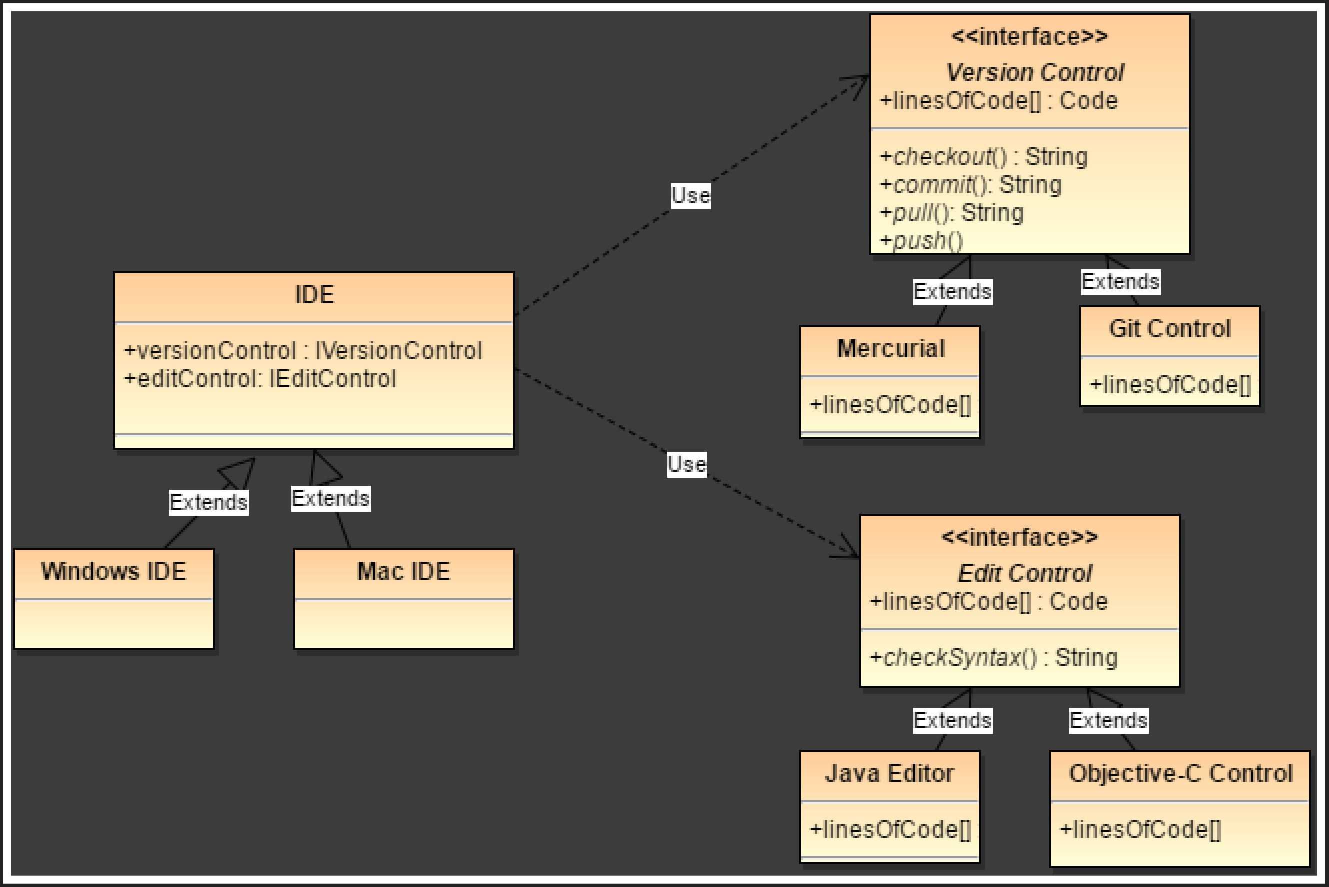
DYNAMIC CHANGE!

But I work on multiple projects, and want to switch between repositories without restarting my IDE!

And ... I want to change programming language

And ... I want the GUI interface to remain the same

AVOID DUPLICATION



DESIGN PRINCIPLES

Identify aspects of your application that vary and separate them from what stays the same

Favour *composition* over *inheritance*

Program to an interface. Delegate actual behaviour to implementing class

PATTERN #1

Strategy Pattern: Defines a family of algorithms/behaviour, encapsulates each one and makes them interchangeable. Strategy lets algorithm vary independently from the client that uses it.

THAT'S ALL, FOLKS!

Questions? Comments?