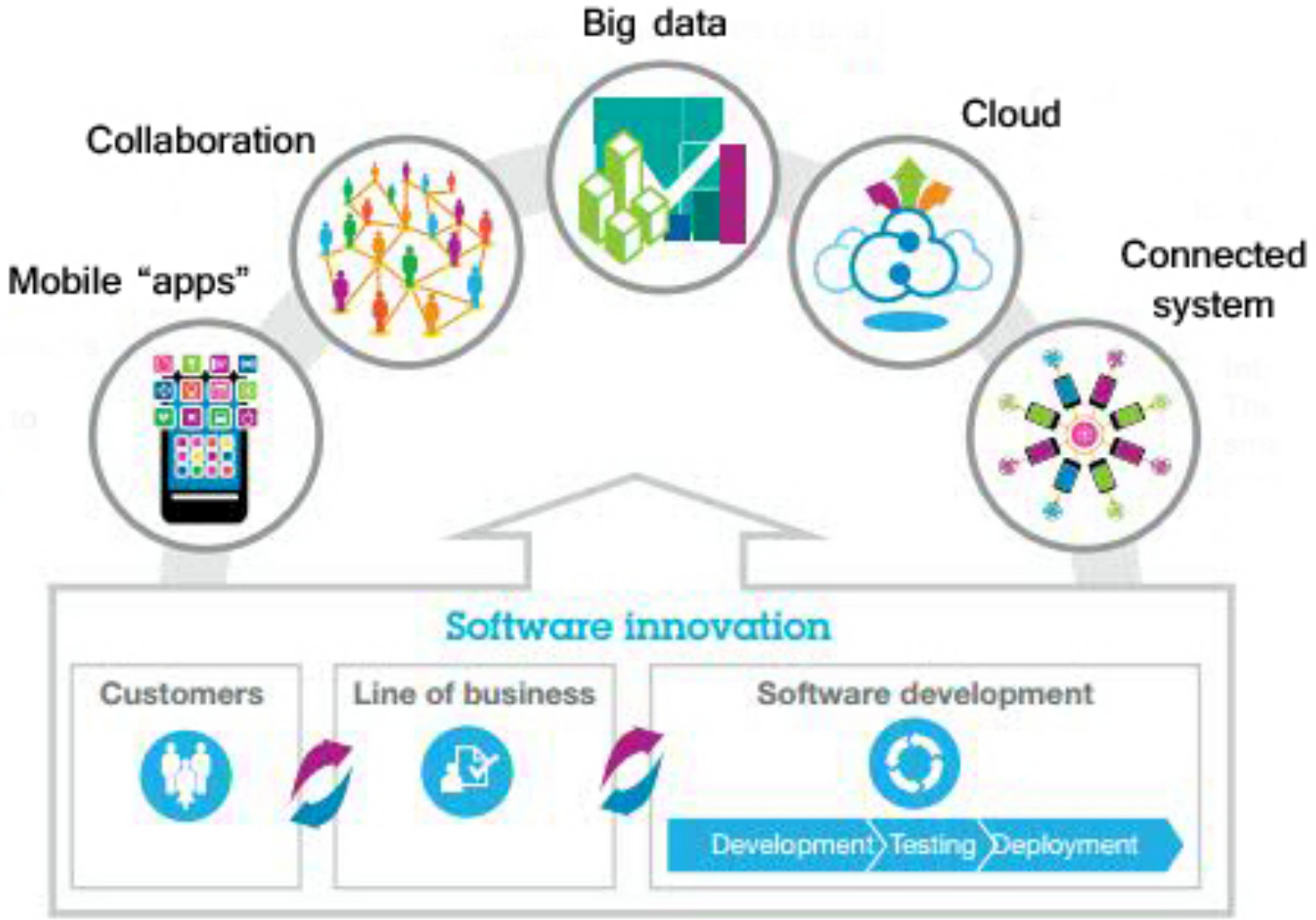
# 301AA - Advanced Programming

Lecturer: **Andrea Corradini**
andrea@di.unipi.it
http://pages.di.unipi.it/corradini/

Course pages:
http://pages.di.unipi.it/corradini/Didattica/AP-18/

***AP-02**: Motivations and Introduction*

# Software is Everywhere



# Programming in the 21 century

- Software as complex as ever
- Command line interface not enough
- Data comes from multiple sources: structured (DB) and unstructured
- Single computer not enough
- Software development is a group activity
- Deployment on Web or mobile devices

# Complexity Prompts for Innovation

- Object-Oriented Programming allows ever larger applications to be built
- But limited support for reuse
- OS + libraries not enough
- Reusable components are needed
- Multi-tier applications development increases the choices on how to build applications

# Key Ingredients for Complex Software

- **Advanced features** extending programming languages
- **Component models** to ensure reusability
- **Frameworks** to support efficient development of (component based) applications
- **Execution environments** providing runtime support for ever dynamic software systems

# The Software Architect

- A new role is needed: **Software Architect**
- to create, define or choose an **application framework**
- to create the component design according to a **component model**
- to structure a complex application into pieces
- to understand the interactions and dependencies among components
- to select the **execution environment / platform** based on cost/performance criteria
- to organize and supervise the development process

# What are Frameworks?

- **Software Framework**: A collection of *common code* providing *generic functionality* that can be *selectively overridden or specialized* by user code providing *specific functionality*
- **Application Framework**: A software framework used to implement the *standard* structure of an application for a *specific* development environment

# Framework Features

- Frameworks, like ***software libraries***, provide ***reusable abstractions*** of code wrapped in a well-defined API
- But: **Inversion of control**
  - unlike in libraries, the overall program's flow of control is not dictated by the caller, but by the framework
- Helps solving recurring design problems
- Drives solution
  - Provides a default behavior
  - Dictates how to fill-in-the-blanks
- Non-modifiable framework code
  - Extensibility: usually by selective overriding

# OO Software Framework

- Object-oriented programming frameworks consists of a **set of abstract classes**
- An application can be built simply inheriting from pre-existing classes in the framework
- Instantiation of a framework consists of composing and subclassing the existing classes

# Examples of Frameworks

- General software frameworks
  - **.NET** – Windows platform. Provides language interoperability
  - **Android SDK** – Supports development of apps in Java (but does not use a JVM!)
  - **Spring** – Cross-platform, for Java applications
  - **Cocoa** – Apple's native OO API for macOS. Includes C standard library and the Objective-C runtime.
  - **Eclipse** – Cross-platform, easily extensible IDE with plugins

# Examples of Frameworks

- Frameworks for Application with GUI
  - **MFC** - Microsoft Foundation Class Library. C++ object-oriented library for Windows.
  - **Gnome** – Written in C; mainly for Linux
  - **Qt** - Cross-platform; written in C++

# Examples of Frameworks

- Web Application Frameworks [based on Model-View-Controller design pattern]
  - **ASP.NET** by Microsoft for web sites, web applications and web services
  - **GWT** - Google Web Toolkit (GWT)
  - **Rails** - Written in Ruby - Provides default structures for databases, web services and web pages.

# Examples of Frameworks

- Concurrency
  - **Hadoop Map/Reduce** - software framework for applications which process big amounts of data in-parallel on large clusters (thousands of nodes) in a fault-tolerant manner.
    - **Map:** Takes input data and converts it into a set of tuples (key/value pairs).
    - **Reduce:** Takes the output from Map and combines the data tuples into a smaller set of tuples.

# Framework Design

- Intellectual Challenging Task
- Requires a deep understanding of the problem domain
- Requires mastering of **software (design) patterns**, OO methods and **polymorphism** in particular

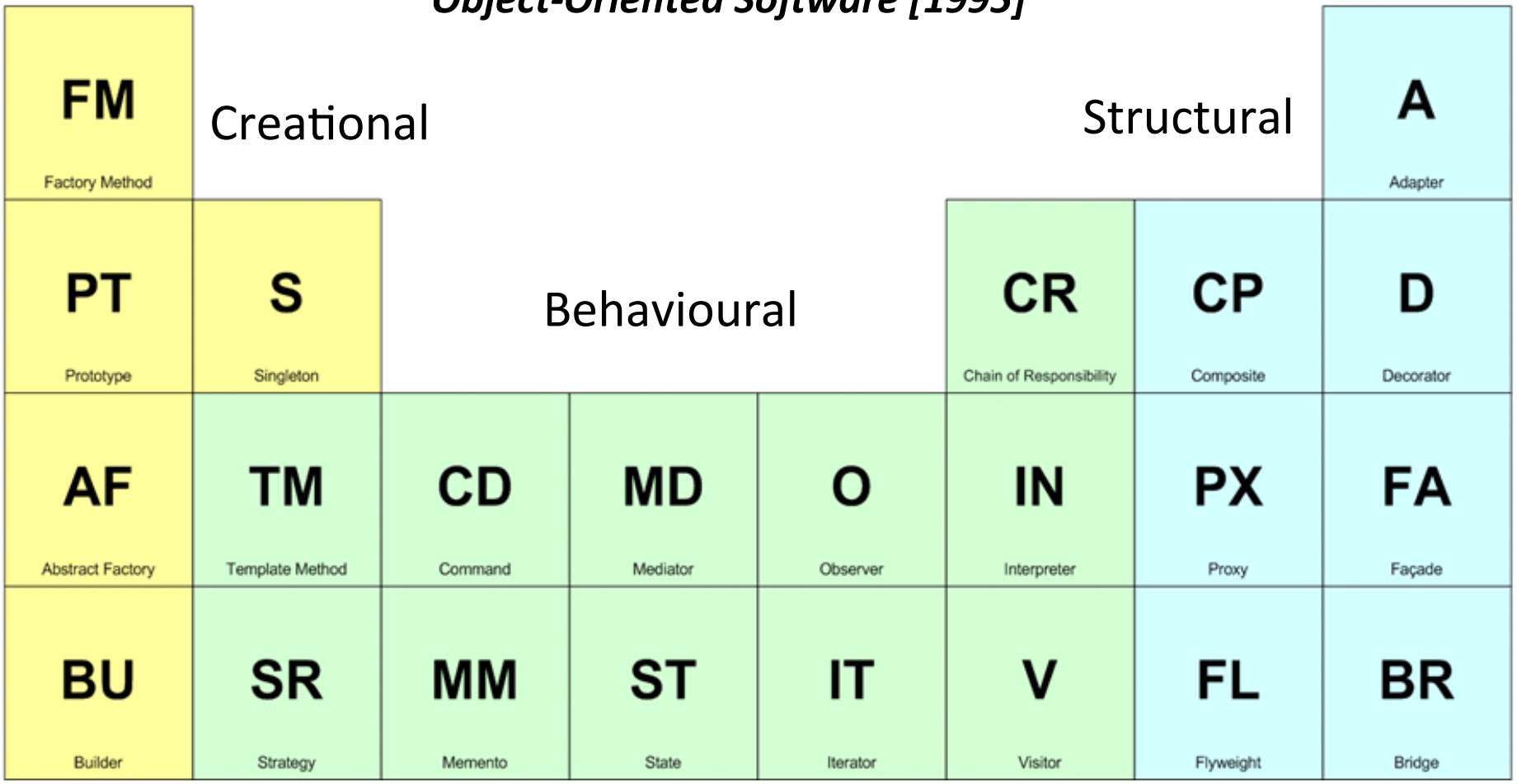
# Design Patterns

- *General conceptual solutions to recurrent design problems*
- *More abstract than frameworks*
  - Frameworks can be embodied in code, but only *examples* of patterns can be embodied in code
  - Design patterns explain the intent, trade-offs, and consequences of a design
- *Smaller architectural elements than frameworks*
  - A typical framework contains several design patterns but the reverse is never true.
- *Less specialized than frameworks*
  - Frameworks always have a particular application domain
  - Design patterns can be used in nearly any kind of application

# The 23 Design Patterns of the Gang of Four

Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides

***Design Patterns: Elements of Reusable Object-Oriented Software [1995]***



# Course objectives and Syllabus

# Course Objectives

- Understand programming language technology:
  - Execution Models
  - Run-time
- Analyze programming metaphors:
  - Objects
  - Components
  - Patterns
- Learn advanced programming techniques
- Present state-of-the-art frameworks incorporating these techniques
- Practice with all these concepts through small projects

# Run-time Systems

- Virtual Execution Environment
  - Memory Management
  - Thread Management
  - Exception Handling
  - Security
  - Debugging Support
  - AOT and JIT Compilation
  - Dynamic Link/Load
  - Reflection
  - Verification
- A concrete example: the JVM

# Component Models and Frameworks

- Component-oriented Programming
- JavaBeans and NetBeans
- Spring and Spring Beans
- COM
- CLR and .NET
- OSGi and Eclipse
- Hadoop Map/Reduce

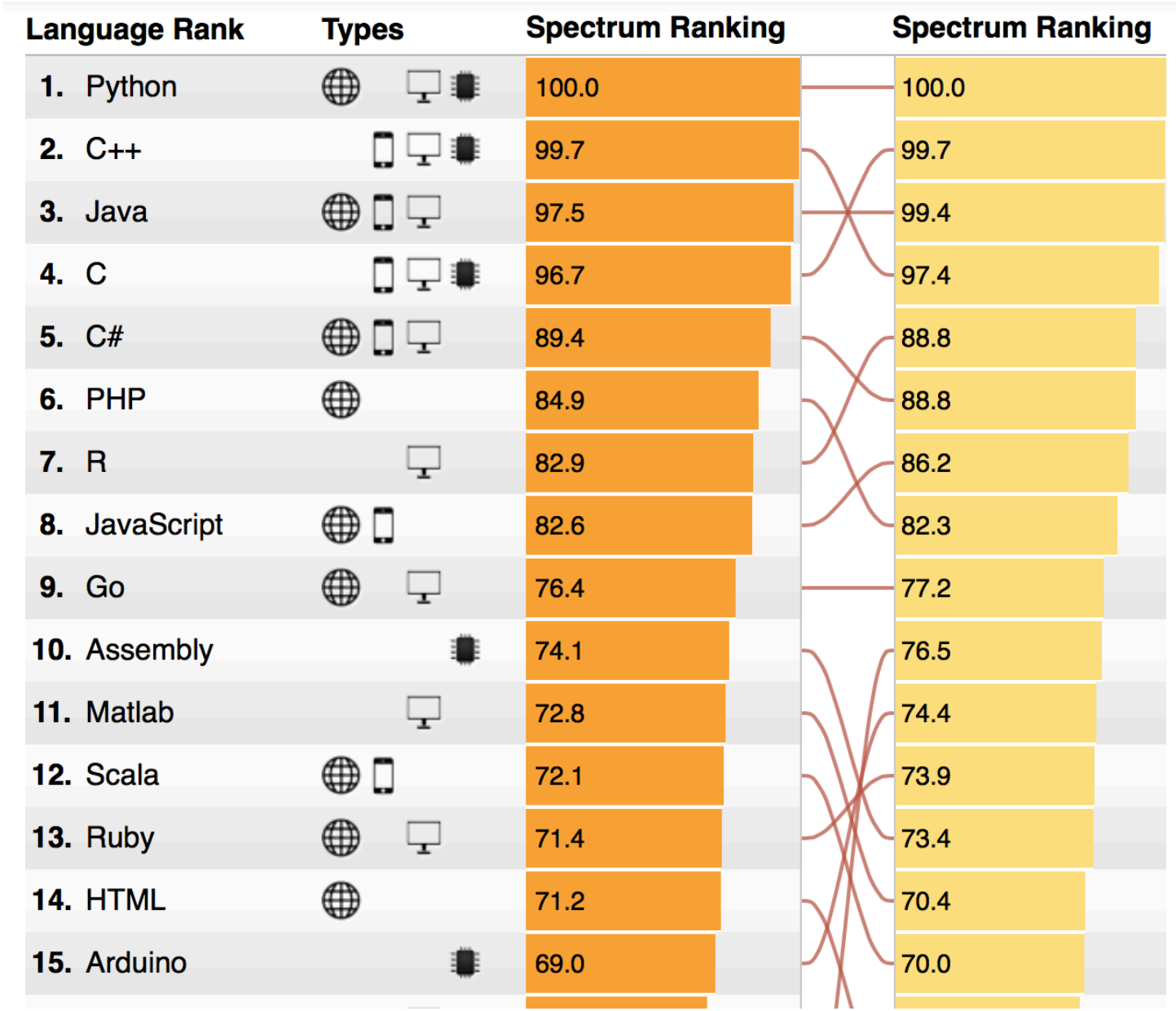
# Advanced Programming Techniques

- Generic Programming
  - Java Generics
  - C++ templates
  - C# Generics
  - Scala generics
- Lambda Calculus and Functional Programming
  - Haskell basics
  - Type classes and Monads
  - Metaprogramming
- Functional Programming in Java 8
  - Lambdas
  - Stream API
- Scripting languages and Python

# Selected Advanced Concepts in Programming Language

- Overloading and Type Classes in Haskell
- Closures vs Delegates in CLI
- Algebraic data types and Active patterns in F#
- Associative arrays in scripting languages
- Ownership and borrowing in Rust
- Extensions in Swift

# IEEE Spectrum Ranking 2018-2017



# IEEE Spectrum Ranking 2017-2016