

# Design Patterns

MSc in Computer Science

---

Produced  
by

Eamonn de Leastar  
edeleastar@wit.ie

Department of Computing, Maths & Physics  
Waterford Institute of Technology

<http://www.wit.ie>

<http://elearning.wit.ie>



Waterford Institute of Technology  
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRCE



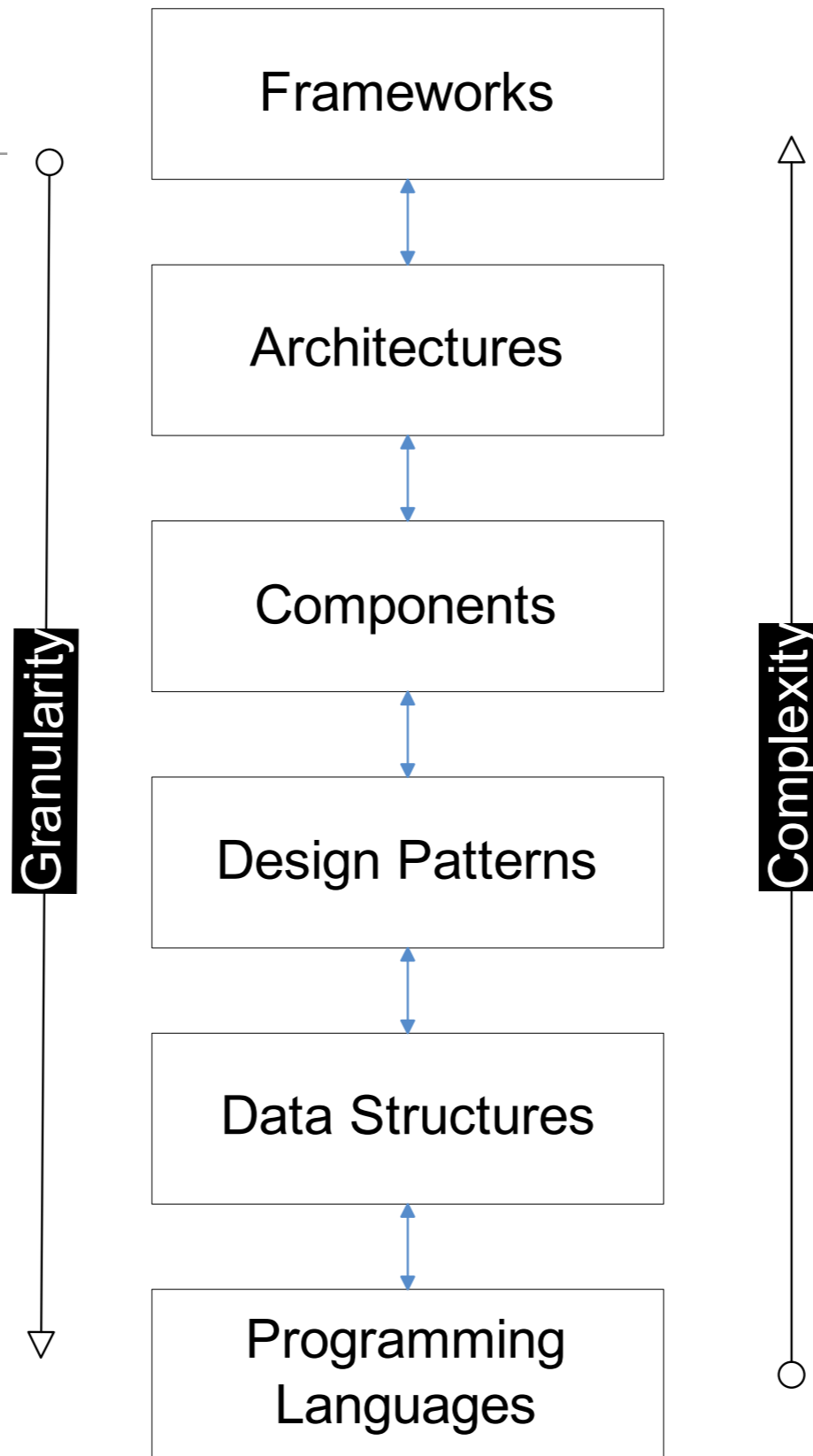
# Design Patterns Context

---

Agile Software Development

# Context

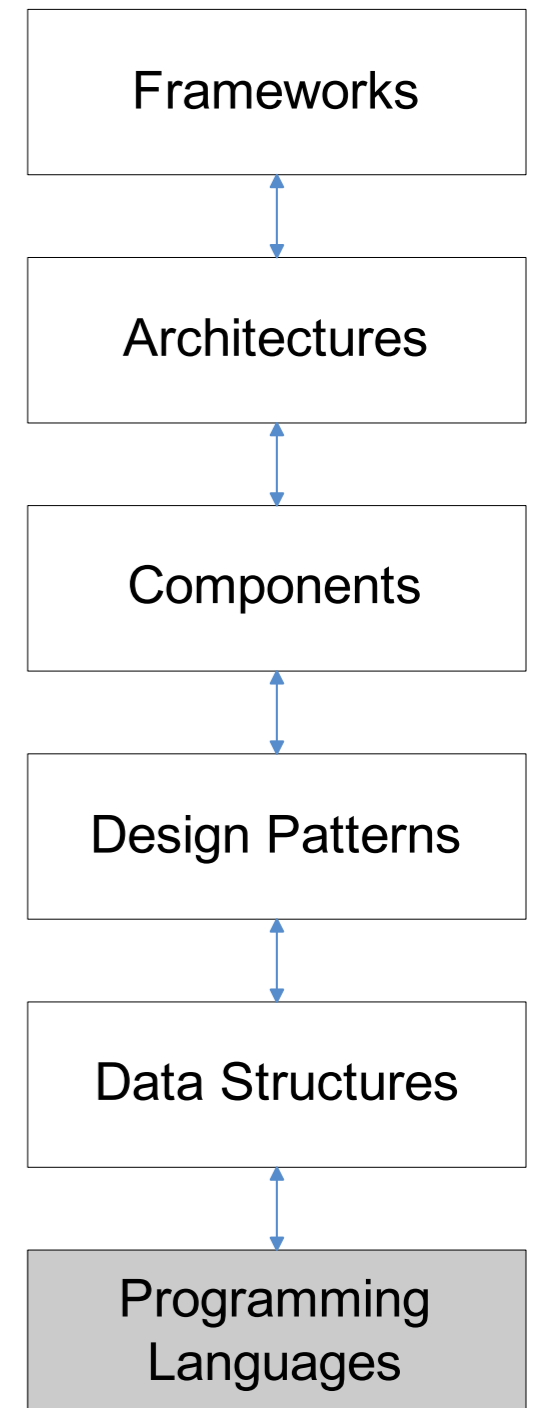
---



# Programming Languages

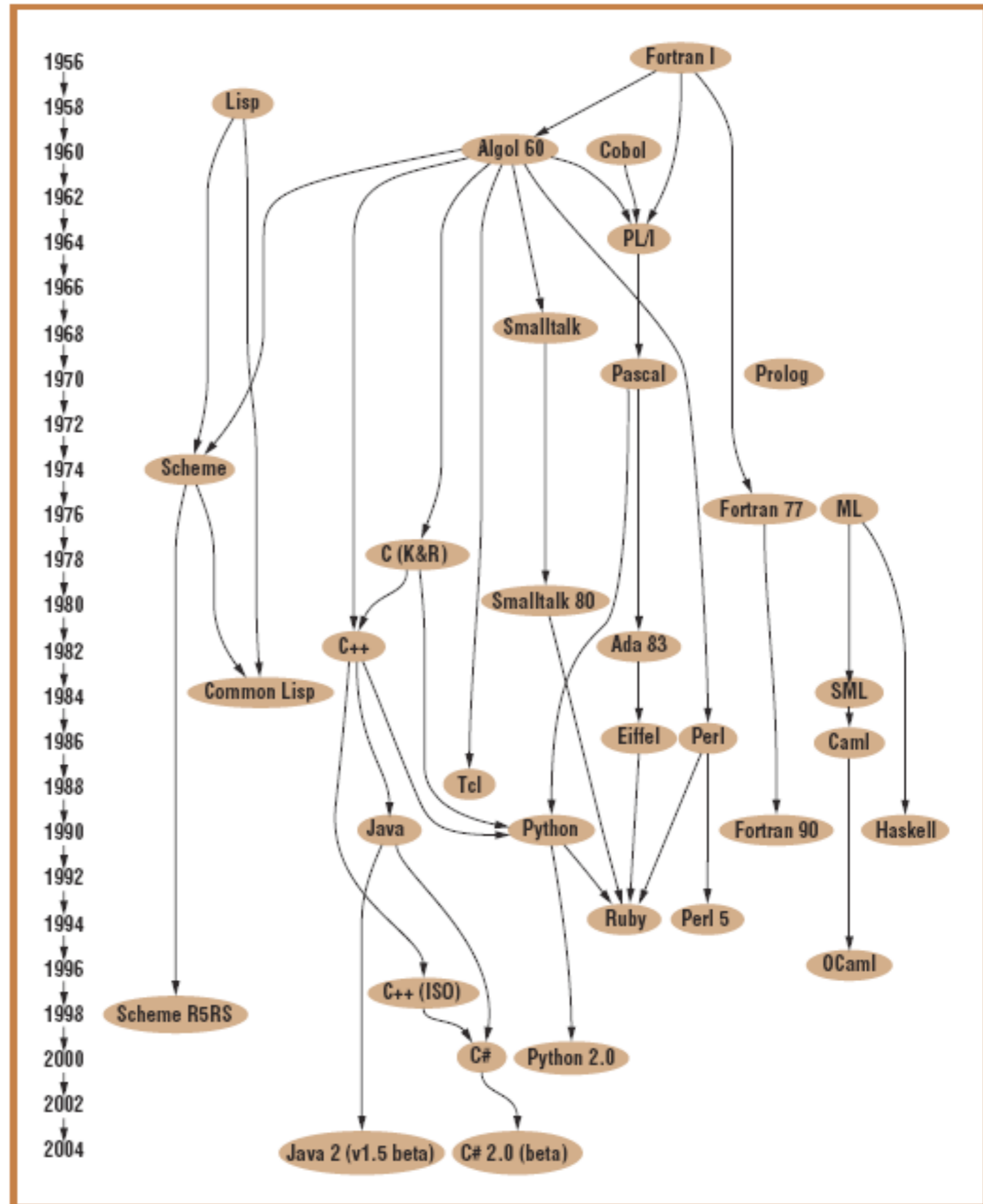
---

- A programming language is a system of signs used to communicate a task/algorithm to a computer, causing the task to be performed
- The task to be performed is called a computation, which follows absolutely precise and unambiguous rules.
- Three components:
  - The syntax of the language is a way of specifying what is legal in the phrase structure of the language; (analogous to knowing how to spell and form sentences English)
  - The second component is semantics, or meaning, of a program in that language.
  - Certain idioms that a programmer needs to know to use the language effectively - are usually acquired through practice and experience



# Family Tree

- Imperative languages: (Fortran, C, and Ada) enable programmers to express algorithms for solving problems.
- Declarative languages, (Lisp, Prolog, Haskell) allow the programmer to specify what has to be computed, but not how the computation is done.
- Object Oriented: can be viewed as a hybrid – of declarative (class structures) & imperative (methods) features.



# Characteristics of OO Languages

---

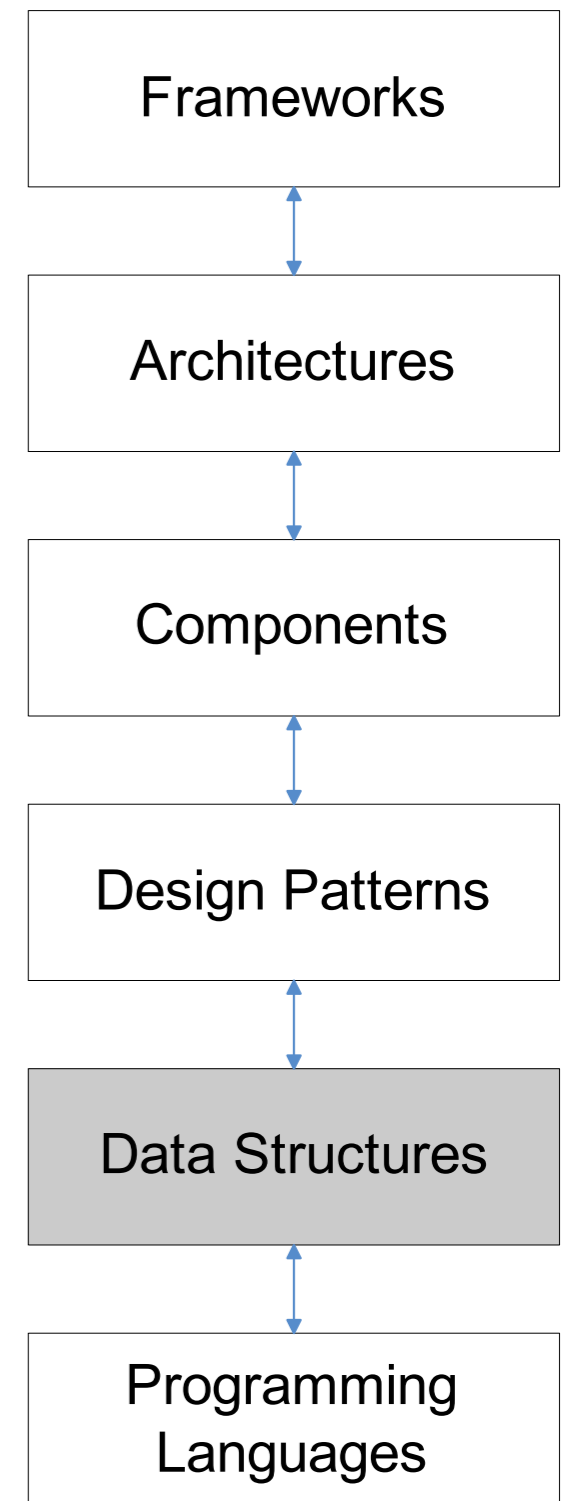
- 1.Object-based modular structure.
- 2.Data abstraction.
- 3Automatic memory management.
- 4.Classes.
- 5.Inheritance.
- 6.Polymorphism and dynamic binding.
- 7.Multiple and repeated inheritance.

(Meyer)

# Data Structures & Problems

---

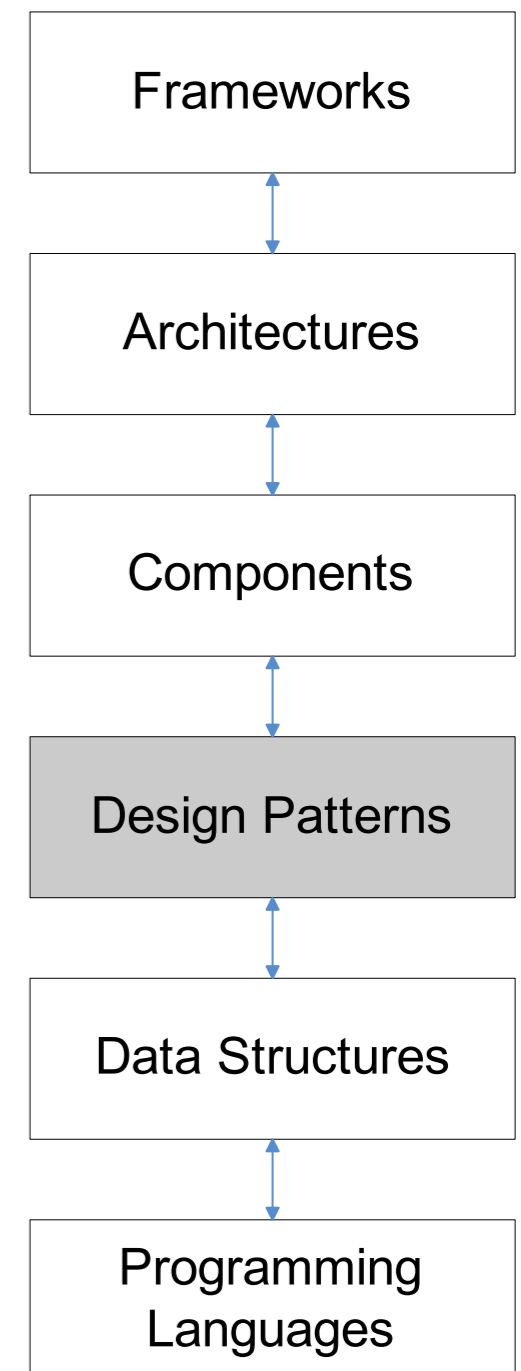
- Typical Data Structures:
  - Lists, Stacks, Queues, Trees, Heaps
  - Static and Dynamic implementations
- Typical Problem Categories:
  - Search
  - Decision
  - Classification
  - Generation & Enumeration
  - Aggregation & Clustering
  - Sorting
  - Traversal



# Design Patterns

---

- A design pattern is a proven solution for a general design problem.
- It consists of communicating ‘objects’ that are customized to solve the problem in a particular context.
- Patterns have their origin in object-oriented programming where they began as collections of objects organized to solve a problem.
- There isn't any fundamental relationship between patterns and objects; it just happens they began there.
- Patterns may have arisen because objects seem so elemental, but the problems we were trying to solve with them were so complex.





# Pattern Levels

---

## Architectural Patterns:

- Expresses a fundamental structural organization or schema for software systems. It provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them.

## Design Patterns:

- Provides a scheme for refining the subsystems or components of a software system, or the relationships between them. It describes commonly recurring structure of communicating components that solves a general design problem within a particular context.

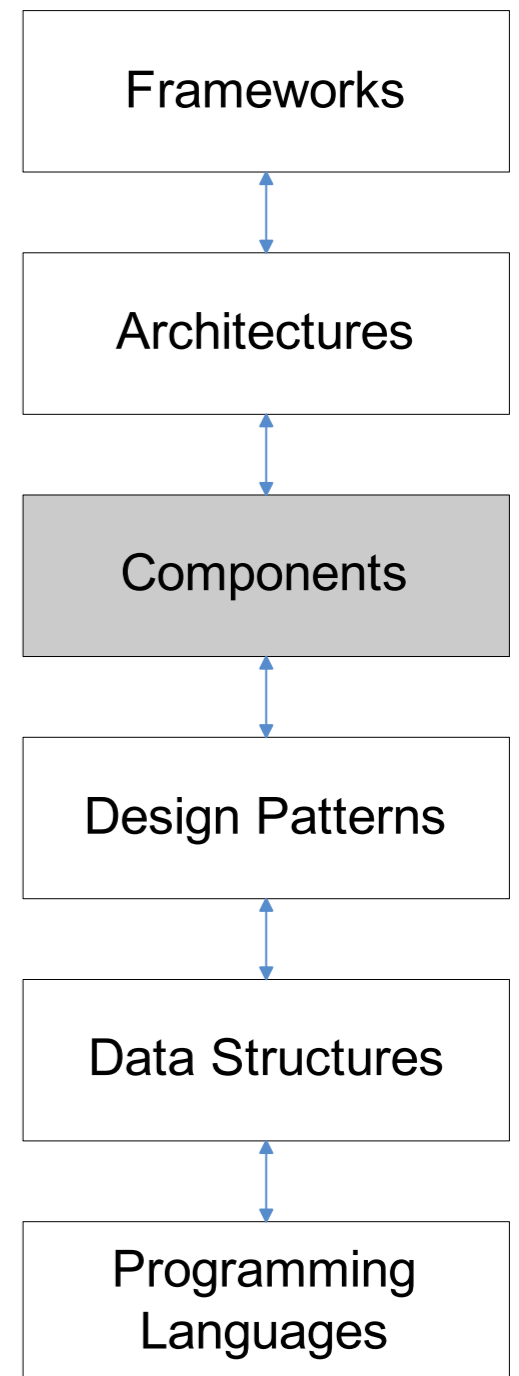
## Idioms:

- A low-level pattern specific to a programming language. An idiom describes how to implement particular aspects of components or the relationships between them using the features of the given language.

# Components

---

- Software components are binary units of:
  - independent production,
  - acquisition,
  - deployment
- that interact to form a functioning program.  
(Szyperski)
- Emphasis has on reusable units
- A component must be compatible and interoperate with a whole range of other components.
- Two main issues arise with respect to interoperability information:
  - How to express interoperability information
  - How to publish this information



# More Component Definitions

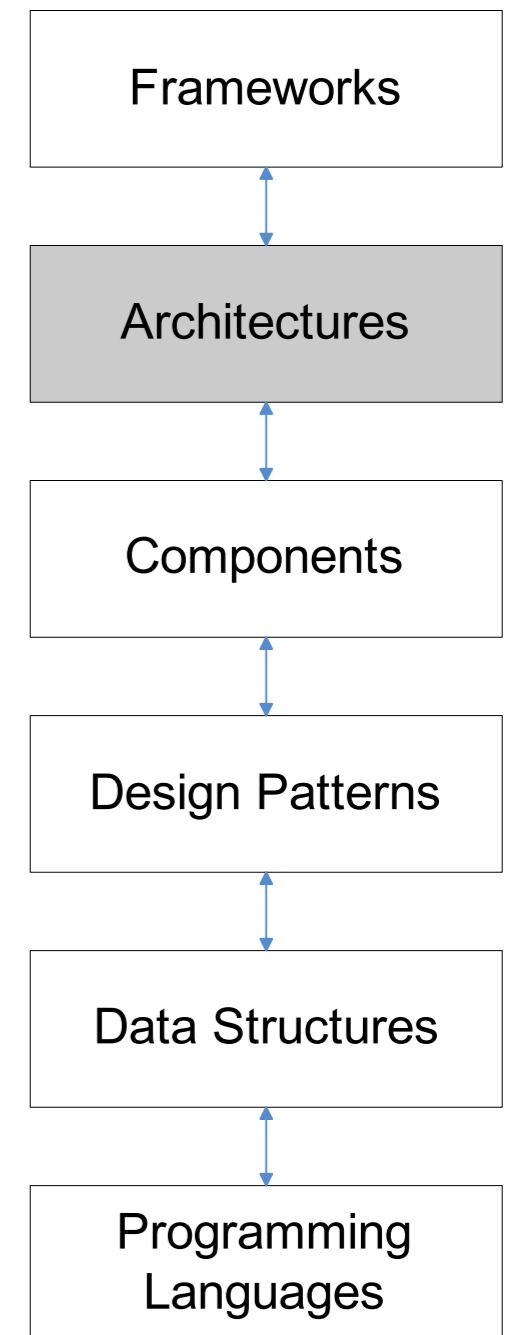
---

- "A component is a nontrivial, nearly independent, and replaceable part of a system that fulfills a clear function in the context of a well-defined architecture. A component conforms to and provides the physical realization of a set of interfaces." (Philippe Krutchen, Rational Software)
- "A runtime software component is a dynamically bindable package of one or more programs managed as a unit and accessed through documented interfaces that can be discovered at runtime." (Gartner Group)
- "A component is a physical and replaceable part of a system that conforms to and provides the realization of a set of interfaces...typically represents the physical packaging of otherwise logical elements, such as classes, interfaces, and collaborations." (Grady Booch, Jim Rumbaugh, Ivar Jacobson, The UML User Guide, p. 343)

# Architecture

---

- The software architecture of a program or computing system is:
  - the structure or structures of the system, which comprise software components,
  - the externally visible properties of those components, and
  - the a set of rules that govern relationships among them.
- An architectural style is a family of software architectures, defining types of components and types of connections, and rules describing how to combine them.
- A software architecture is an instantiation of an architectural style for a certain system. The components and connections may be decomposed into architectures themselves.



# Architectural Styles

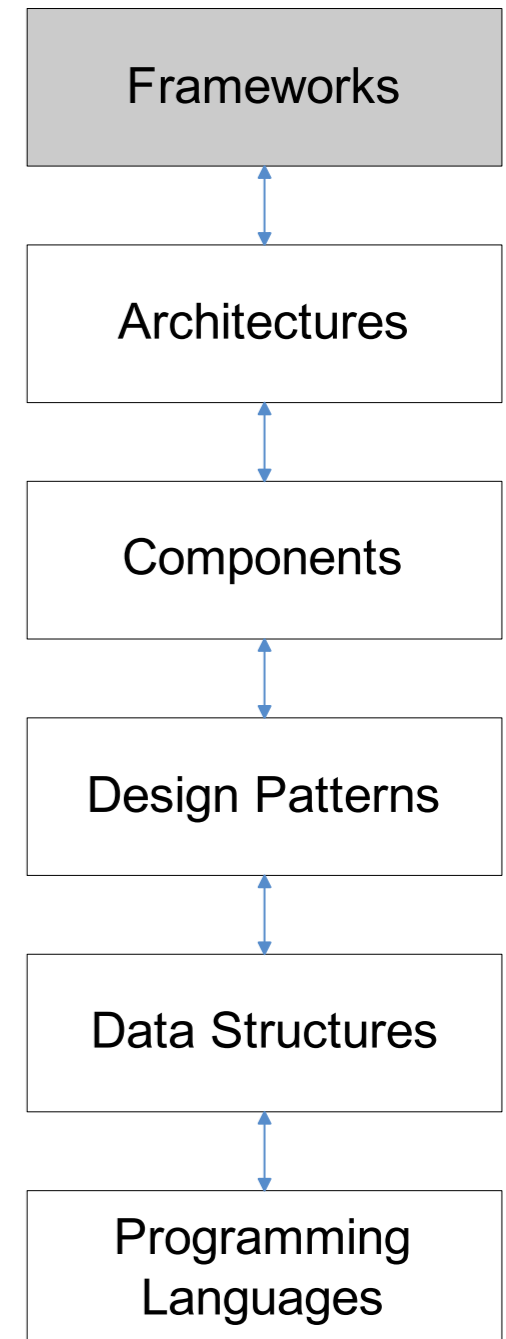
---

- Batch
- Pipe & Filter
- Client/Server
- Blackboard
- Event Driven
- Plug-in
- Space Based (Tuples)
- Three-Tier
- Network
  - Data Flow
  - Replication
  - Hierarchical
  - Mobile Code
  - Peer to Peer

# Frameworks

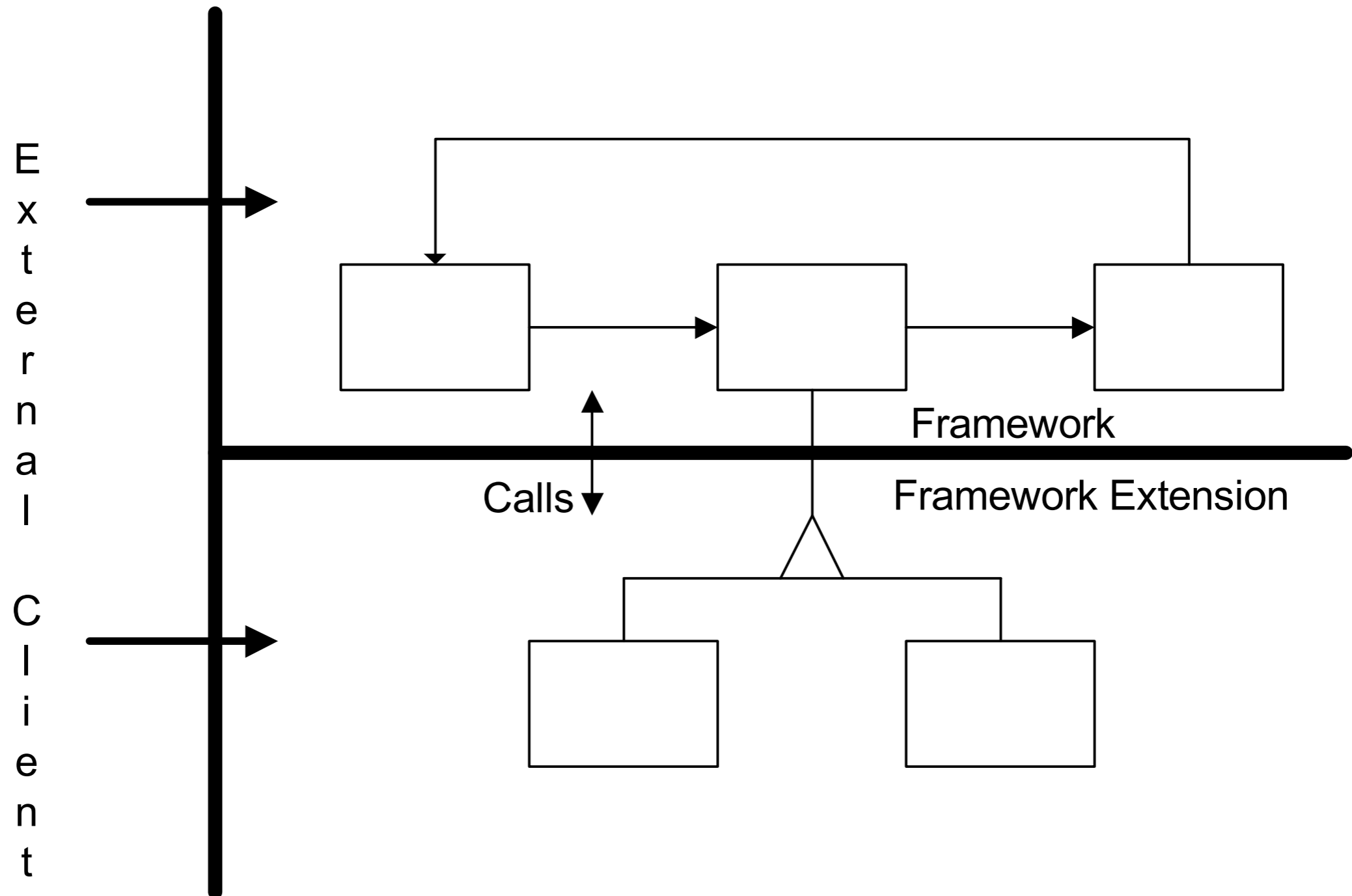
---

- A framework is a set of related components which you specialize, integrate and/or instantiate to implement an application or subsystem
  - Usually, a semi complete application containing dynamic and static components that can be customized to produce applications
- Frameworks are targeted for a particular application domain & consists of a set of classes (abstract & concrete), whose instances:
  - collaborate
  - are intended to be extended, i.e. reused (abstract design)
  - do not have to address a complete application domain (allowing for composition of frameworks)
- Emphasize stable parts of the domain and their relationships and interactions



# Framework Structure

---



# Framework Example

- ▼ > spacebook-p2 [spacebook-p2 master]
  - ▼ app
    - ▶ (default package)
    - ▶ controllers
    - ▶ models
    - ▶ views
    - ▶ views.nav
  - ▼ test
    - ▼ (default package)
      - ▶ ApplicationTest.java
      - ▶ FriendsTest.java
      - ▶ IntegrationTest.java
      - ▶ MessagesTest.java
      - ▶ UsersTest.java
  - ▶ Referenced Libraries
  - ▶ JRE System Library [Java SE 7 [1.7.0\_21]]
  - ▶ > conf
  - ▶ logs
  - ▶ > project
  - ▶ public
  - ▶ target
  - ▶ README



The screenshot shows the Play Framework website homepage. At the top left is the Play logo. To its right are navigation links for "Download", "Documentation", and "Get Involved". Below the navigation is a green banner with the text "The High Velocity Web Framework For Java and Scala" and a play button icon. Below the banner is a video player titled "Introduction to Play Framework for Java developers" with a play button and a "19:28" duration. To the left of the video player is a "GET THE LATEST PACKAGE" section with a "Download 2.1.4" button and a link to "browse all versions". Below that is a "GETTING STARTED WITH" section with a "Java & Scala" button and a link to "read full documentation". At the bottom of the page is a large heading: "Play Framework makes it easy to build web applications with Java & Scala." followed by two paragraphs of text describing the framework's architecture and performance.

play

Download Documentation Get Involved

The High Velocity Web Framework For Java and Scala

Introduction to Play Framework for Java developers

GET THE LATEST PACKAGE

Download 2.1.4

or [browse all versions](#)

GETTING STARTED WITH

Java & Scala

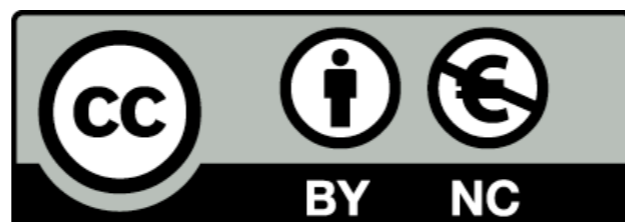
or [read full documentation](#)

Play Framework makes it easy to build web applications with Java & Scala.

Play is based on a lightweight, stateless, web-friendly architecture.

Built on Akka, Play provides predictable and minimal resource consumption (CPU, memory, threads) for highly-scalable applications.





Except where otherwise noted, this content is licensed under a Creative Commons Attribution-NonCommercial 3.0 License.

For more information, please see <http://creativecommons.org/licenses/by-nc/3.0/>

