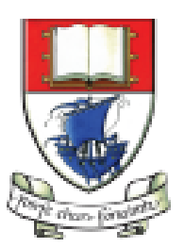
Software Paradigms

Produced by: Eamonn de Leastar (edeleastar@wit.ie)

Dr. Siobhan Drohan (sdrohan@wit.ie)



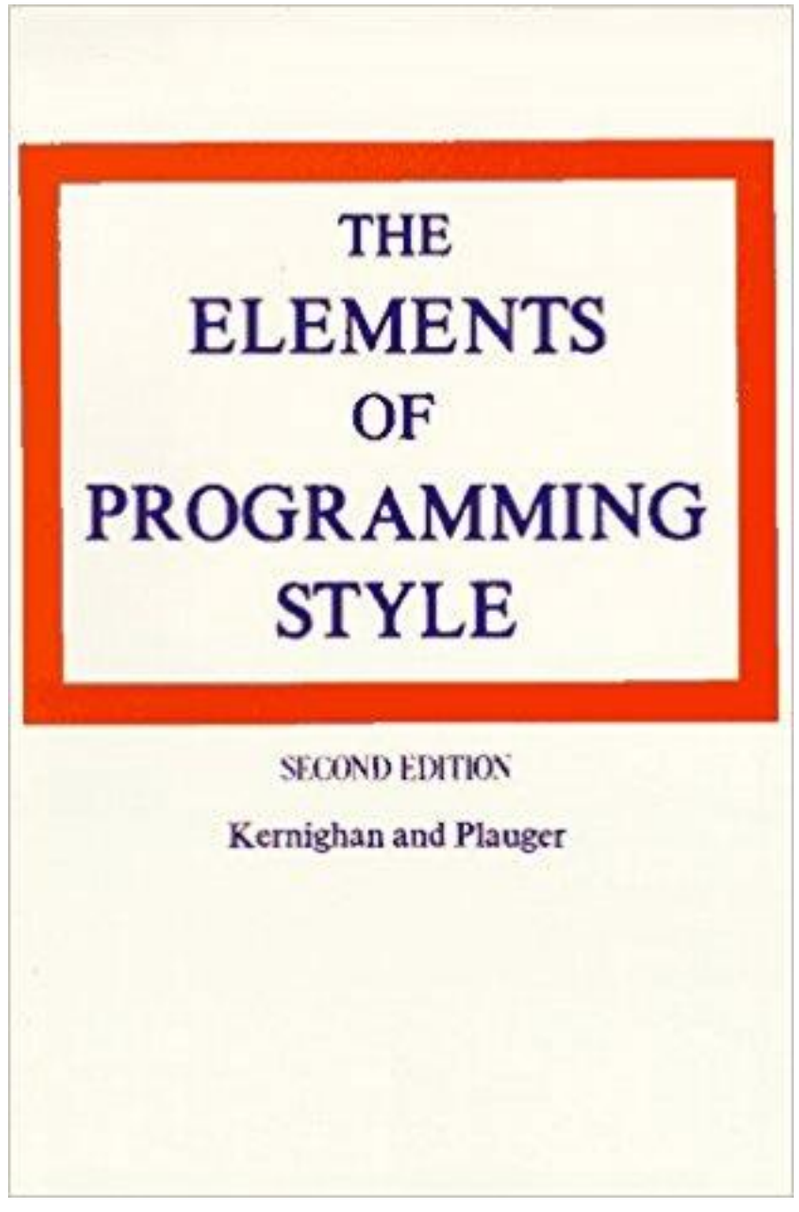
Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

Department of Computing and Mathematics
http://www.wit.ie/

Literate Programmer…

“Good design and programming is not learned by generalities, but by seeing how significant programs can be made clean, easy to read, easy to maintain and modify, human-engineered, efficient, and reliable, by the application of good design and programming practices. Careful study and imitation of good designs and programs significantly improves development skills."

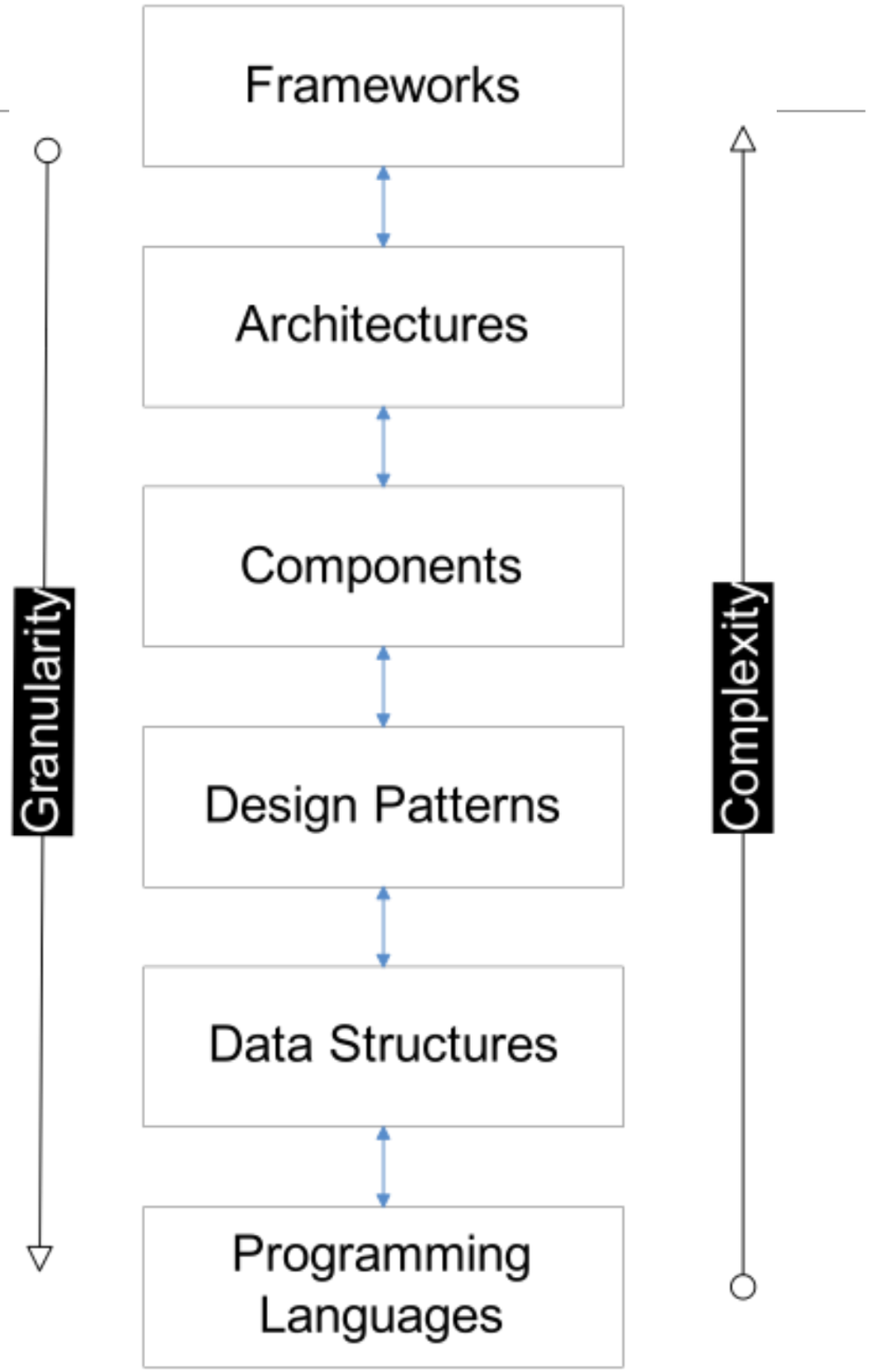
Kernighan & Plauger (1978)



Agenda

Software & Programming Paradigms

Software Engineering Knowledge Stack

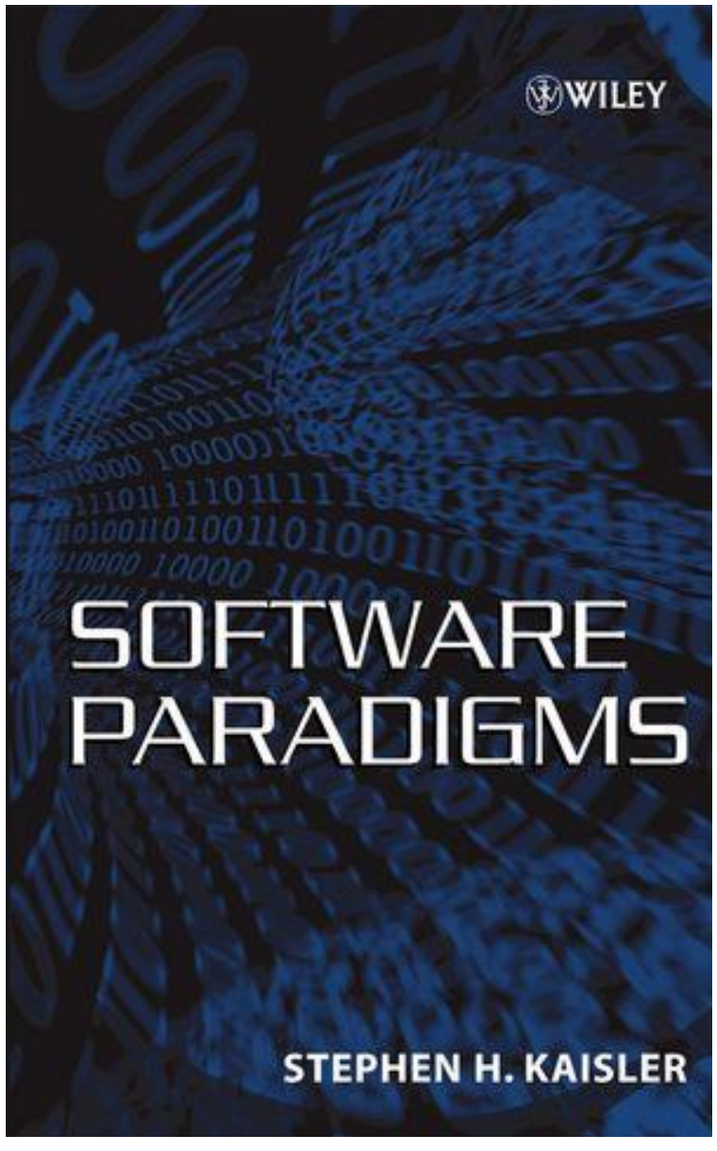


Software Paradigms…

- *Paradigm** is commonly used to refer to a category of entities that share a common characteristic.

- Taken to mean a conceptual way of describing something

- The rate of change in the software discipline has seen proliferation of overlapping paradigms.



*from Greek *paradeigma*, ‘show side by side’,

Software Paradigms (examples..)

- **Imperative programming** – defines computation as statements that change a program state.
- **Procedural programming, structured programming** – specifies the steps a program must take to reach a desired state.
- **Declarative programming** – defines program logic, but not detailed control flow.
- **Functional programming** – treats programs as evaluating mathematical functions and avoids state and mutable data
- **Object-oriented programming (OOP)** – organizes programs as objects: data structures consisting of datafields and methods together with their interactions.
- **Event-driven programming** – program control flow is determined by events, such as sensor inputs or user actions (mouse clicks, key presses) or messages from other programs or threads.
- **Automata-based programming** – a program, or part, is treated as a model of a finite state machine or any other formal automaton.

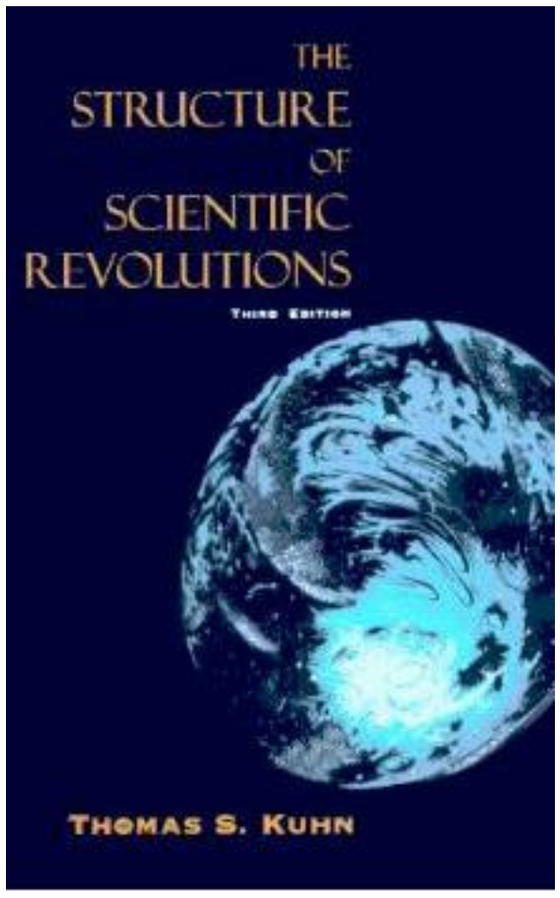
Paradigm Structure

- Two aspects to a paradigm:

  1: Principles and techniques:

  - Symbolic generalizations: Assertions that are later taken for granted and employed without question
  - Model beliefs: a commitment to a belief in a model to which the relevant domain conforms
  - Values

2: Exemplars: shared examples that illustrate the properties of the paradigm.



(Kuhn, “The Structure of Scientific Revolutions”)

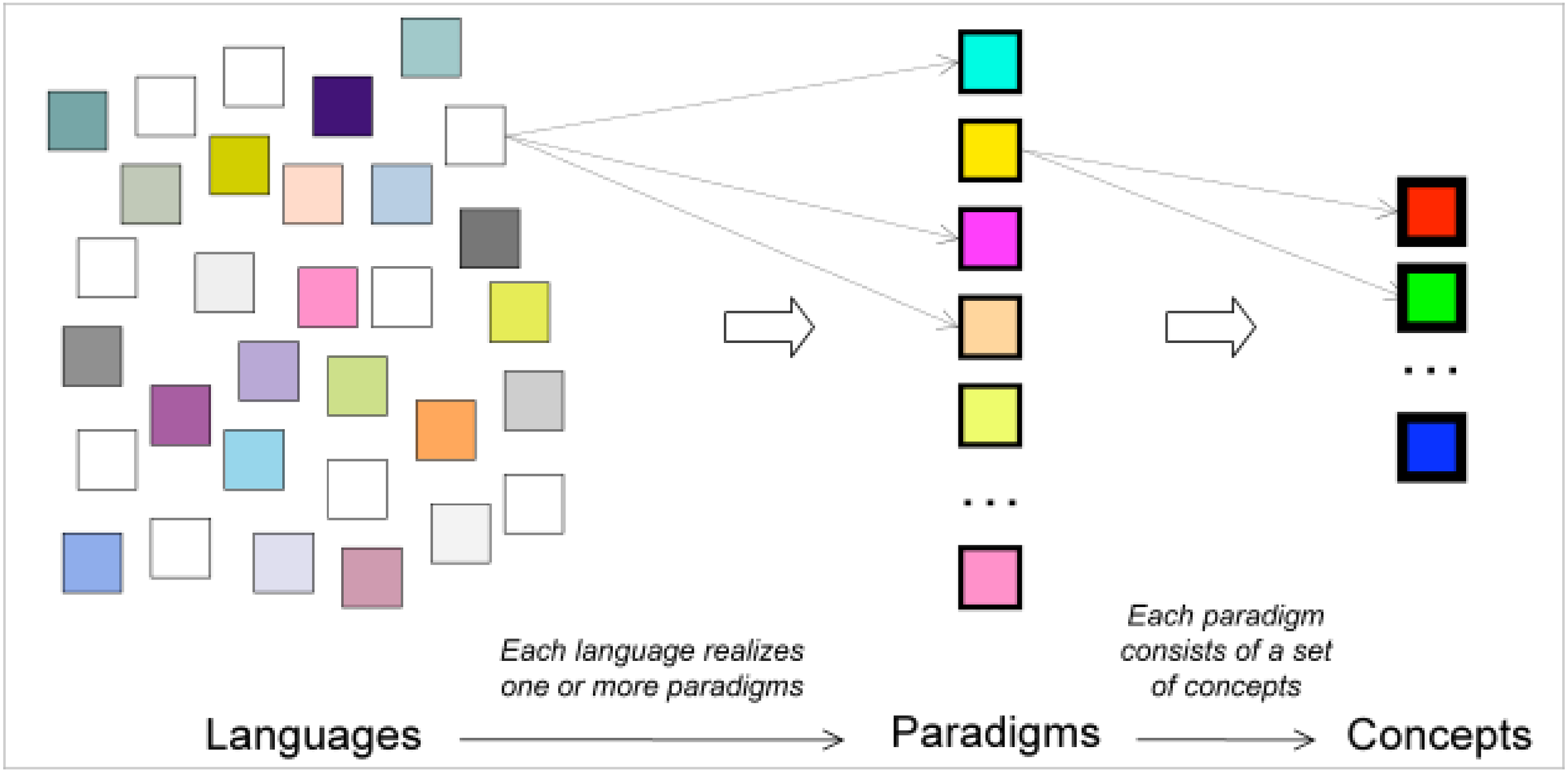
http://en.wikipedia.org/wiki/The_Structure_of_Scientific_Revolutions

x

A fundamental change in the basic concepts and experimental practices of a scientific discipline.

Can be a period of confusion & uncertainty

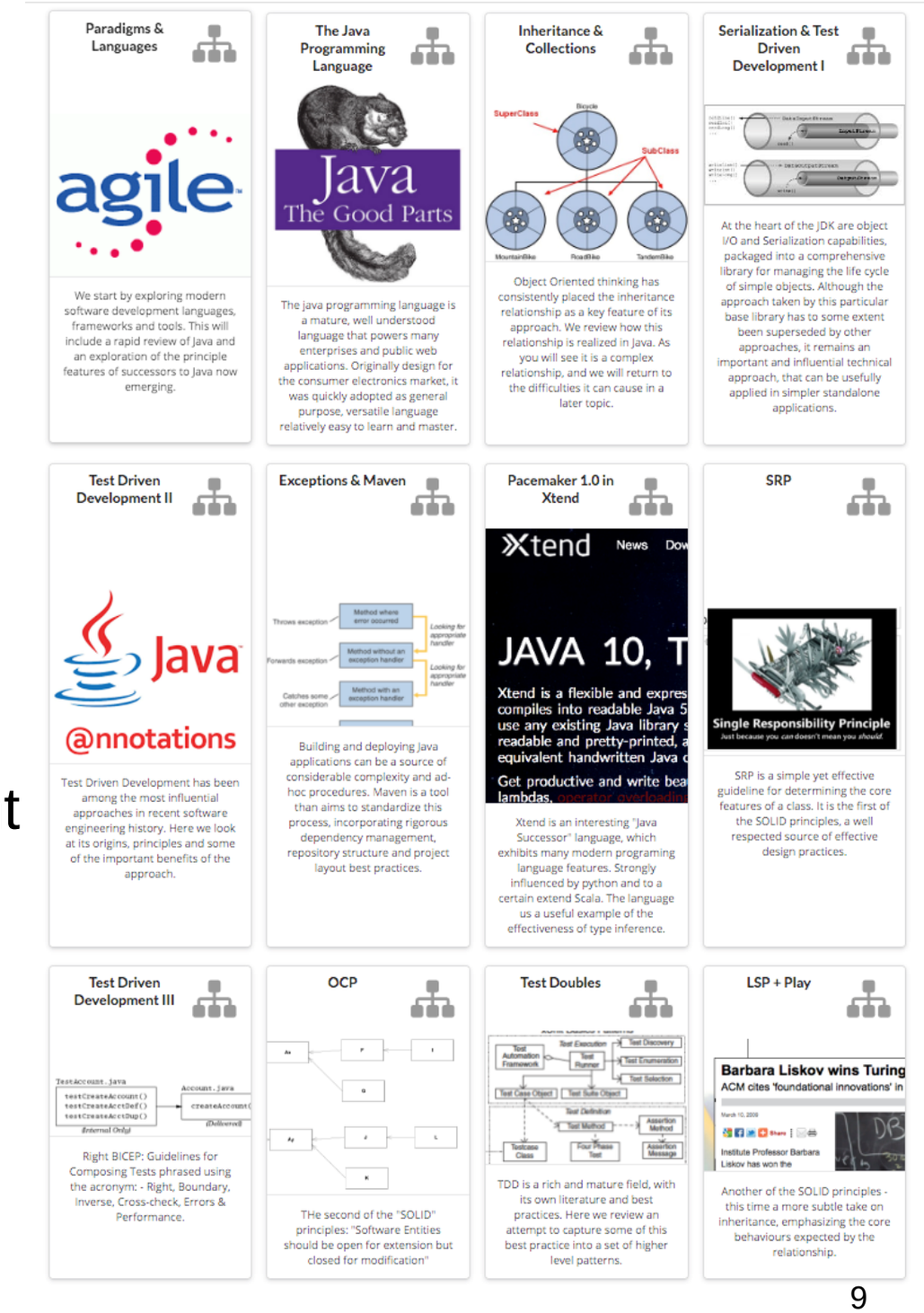
Software Paradigms



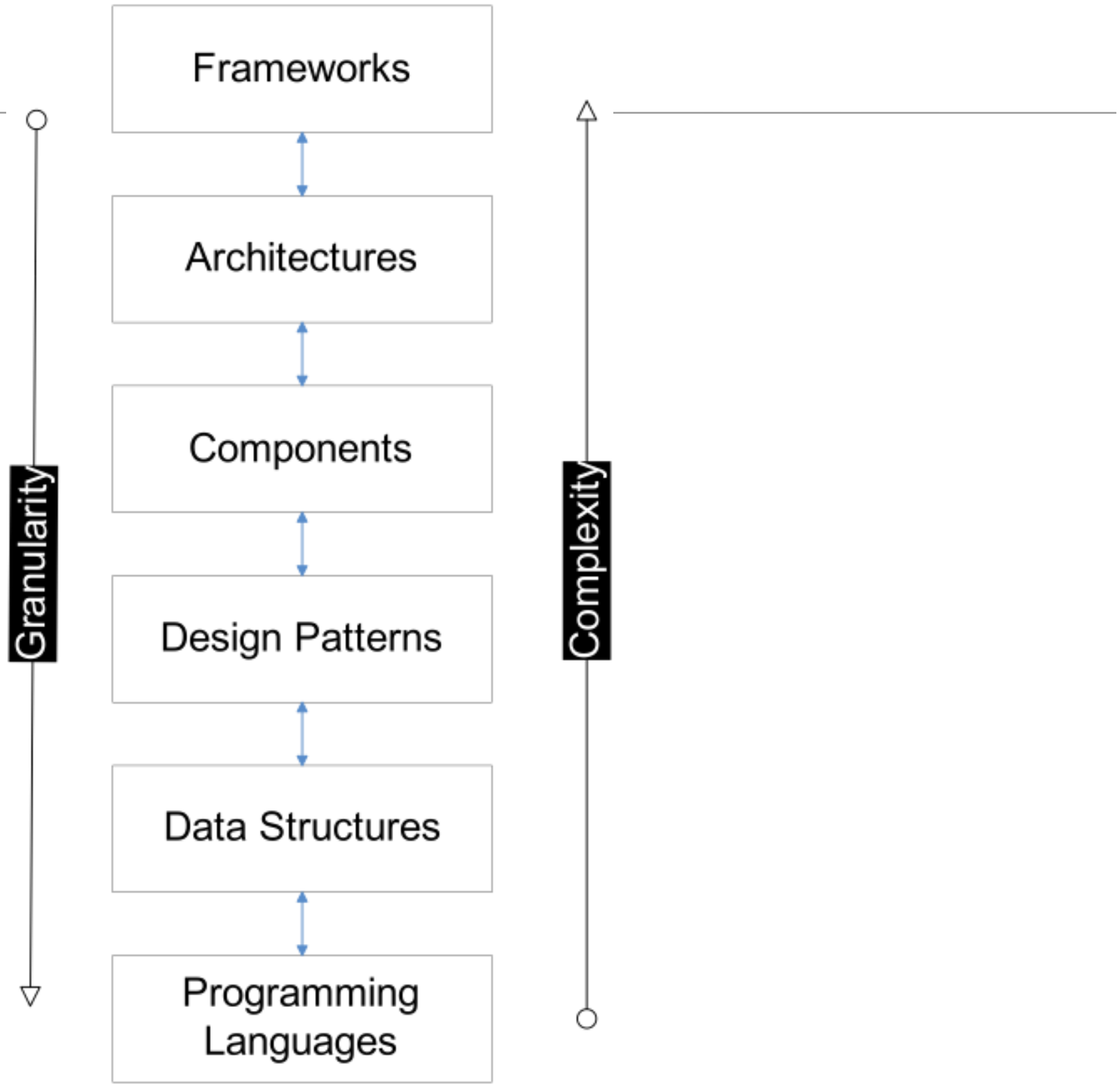
Roy, Peter van (2009)

Paradigms for This Module

- **Object Oriented Programming**
  - OO Principles (particularly SOLID principles)
  - Java Programming Language
  - Kotlin Programming Language
- **Agile Methods**
  - Test Driven Development (TDD)
  - Automated Build / Configuration Management
- **Network Programming**
  - Web Services/HTTP/REST

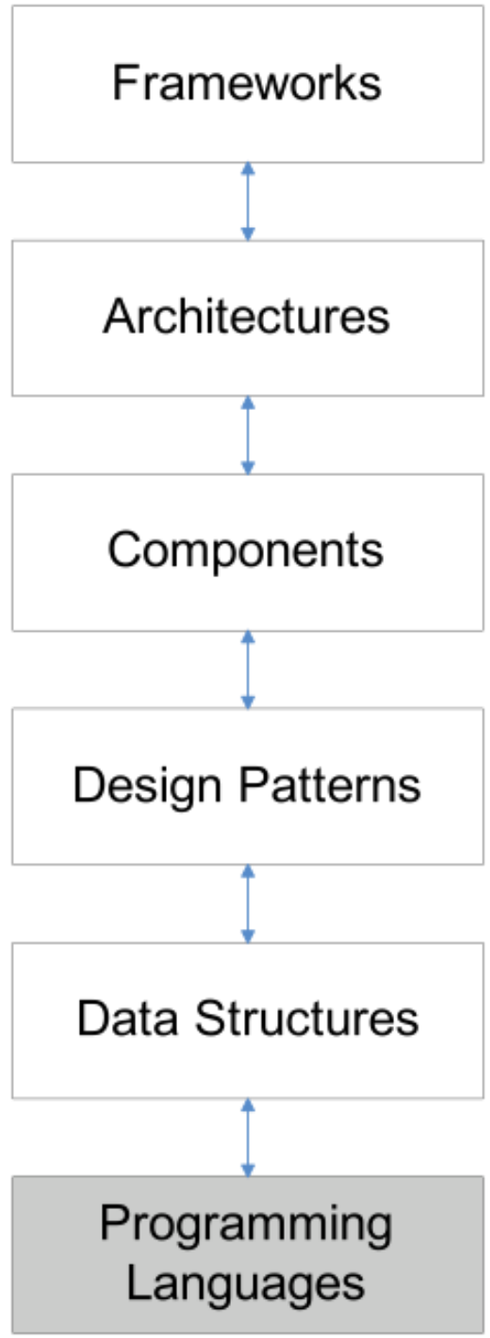


Knowledge Context



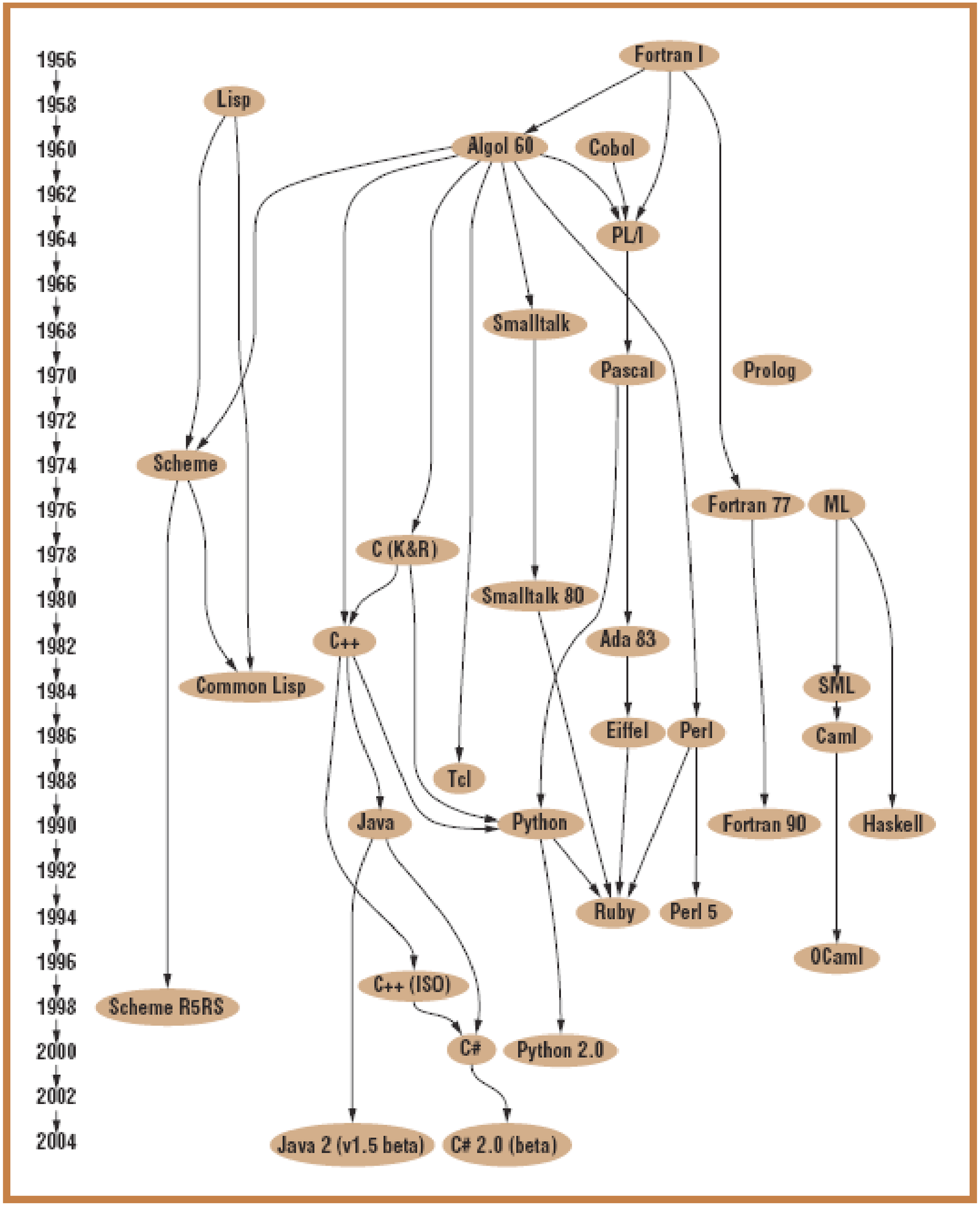
Programming Languages

- 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.



WEB DEVELOPMENT

PHP · C · JAVA SCRIPT · C++ · JAVA · PYTHON · RUBY

GAME DEVELOPMENT

C# · C · C++ · JAVA · PYTHON · RUBY

MOBILE APP DEVELOPMENT

C# · C++ · JAVA

DATA ANALYSIS

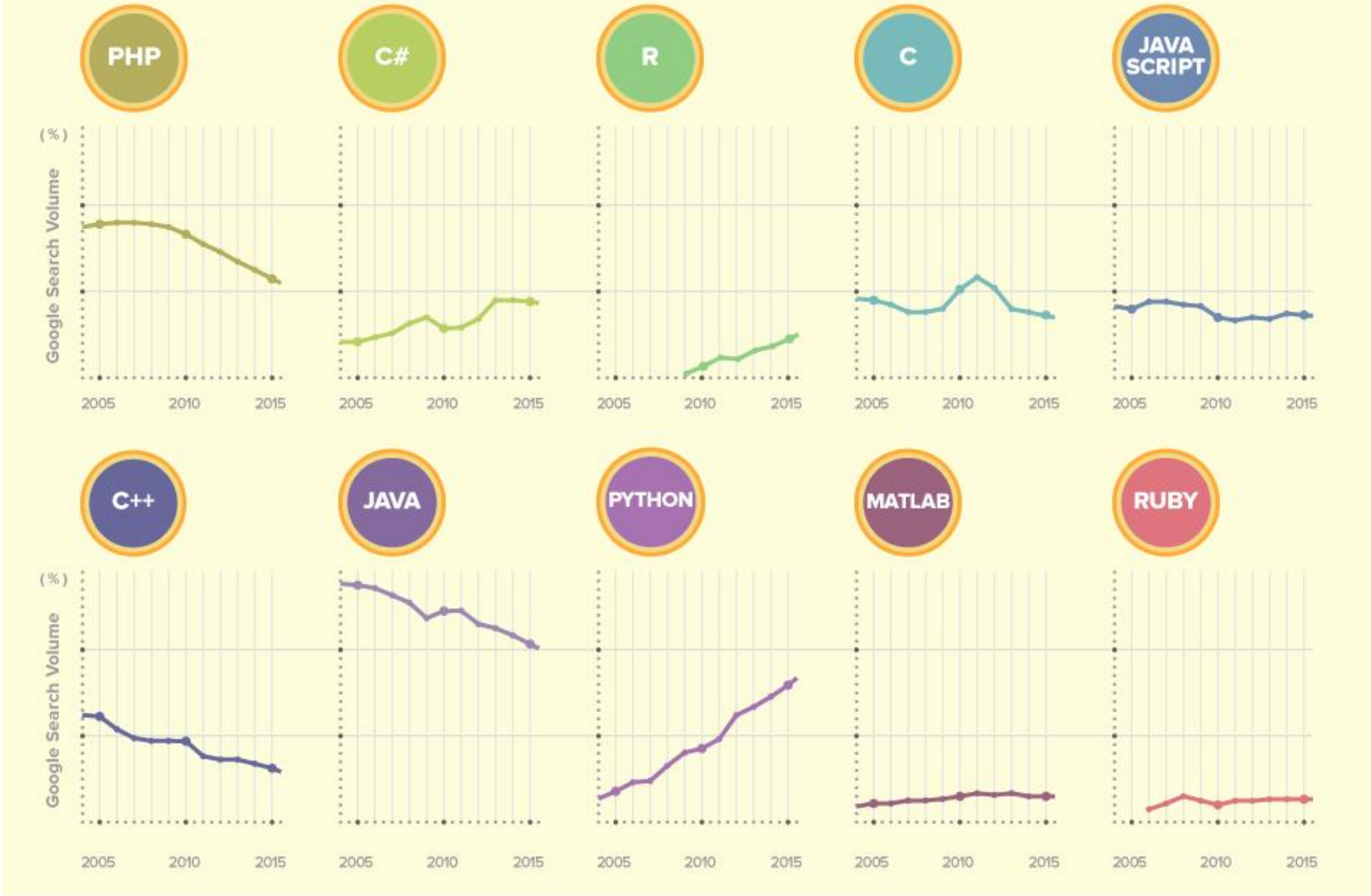
R · PYTHON · MATLAB

EMBEDDED SYSTEM PROGRAMMING

C · C++ · PYTHON

2015 Rank		2015	Change%	2014	Change%	2013	Change%
1	Python	26.67%	-14.64%	31.24%	3.10%	30.30%	5.21%
2	Java	22.58%	15.37%	19.57%	-11.85%	22.20%	-13.95%
3	C++	9.96%	1.76%	9.79%	-24.70%	13.00%	3.17%
4	C#	9.39%	27.37%	7.37%	47.37%	5.00%	100.00%
5	C	7.37%	21.37%	6.07%	48.14%	4.10%	-16.33%
6	JavaScript	6.88%	6.09%	6.48%	24.66%	5.20%	33.33%
7	Ruby	5.88%	-17.27%	7.11%	-32.90%	10.60%	10.42%
8	PHP	3.82%	5.45%	3.62%	9.84%	3.30%	-54.79%
9	Haskell	1.77%	17.24%	1.51%	25.83%	1.20%	
10	Go	1.27%	-44.00%	2.26%	50.67%	1.50%	-25.00%
11	Scala	1.04%	-17.80%	1.27%	27.00%	1.00%	66.67%
12	Perl	0.95%	-37.33%	1.52%	-6.17%	1.62%	
13	Objective-C	0.82%	-17.62%	1.00%	265.76%	0.27%	173.40%
14	Bash	0.46%	7.21%	0.43%	290.91%	0.11%	
15	R	0.37%	165.71%	0.14%	-30.00%	0.20%	
16	Visual Basic,NET	0.37%	825.50%	0.04%			
17	Lua	0.19%	-44.51%	0.35%	337.50%	0.08%	
18	Clojure	0.14%	-8.53%	0.15%	-48.28%	0.29%	-63.75%
19	Tcl	0.06%	-8.57%	0.07%	133.33%	0.03%	50.00%

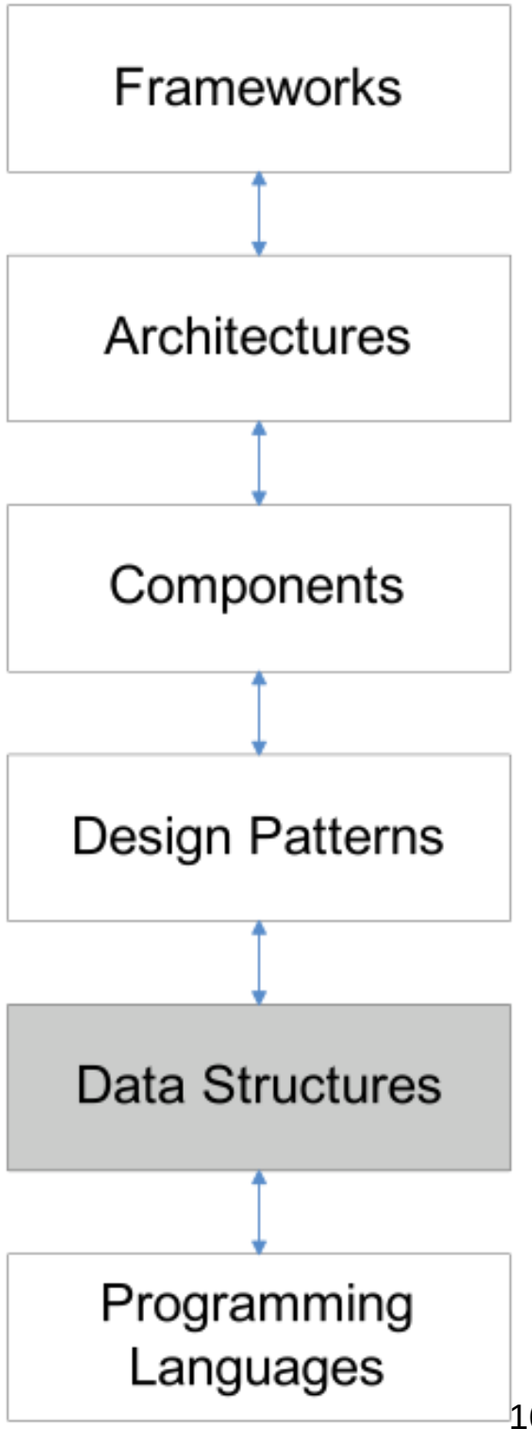
http://blog.codeeval.com/codeevalblog/2016/2/2/most-popular-coding-languages-of-2016



https://blog.udacity.com/2015/05/pick-your-first-programming-language.html

Data Structures & Problems

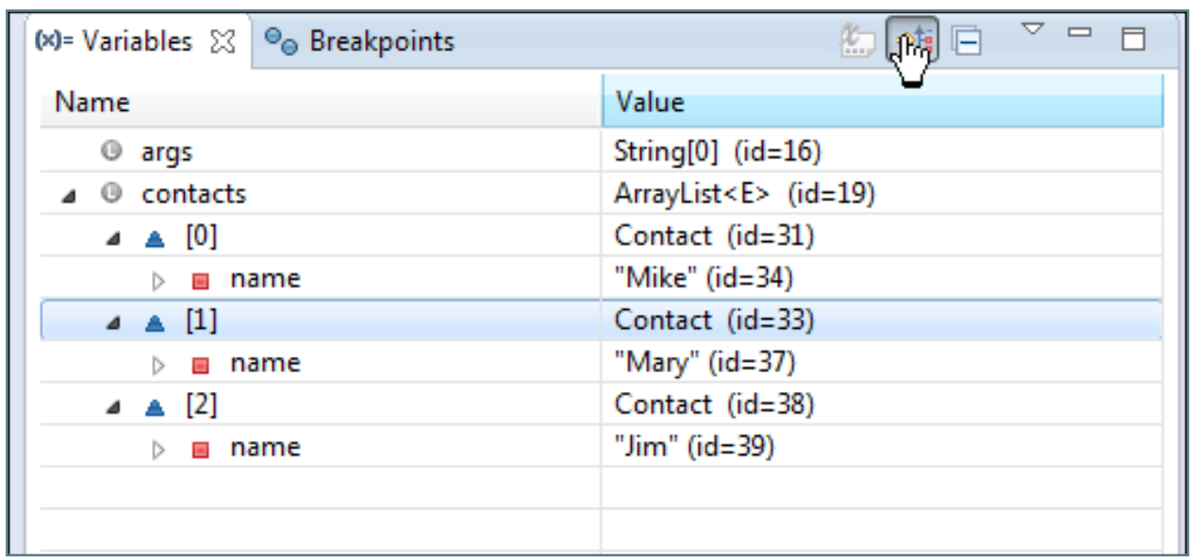
- Typical Data Structures:
  - Lists, Maps, Stacks, Queues, Trees, etc.
  - Static and Dynamic implementations
- Typical Problem Categories:
  - Search
  - Sorting
  - Traversal
  - Inserting / Deleting
  - Merging
  - Clustering
  - Classification



Exploring a Data Structure

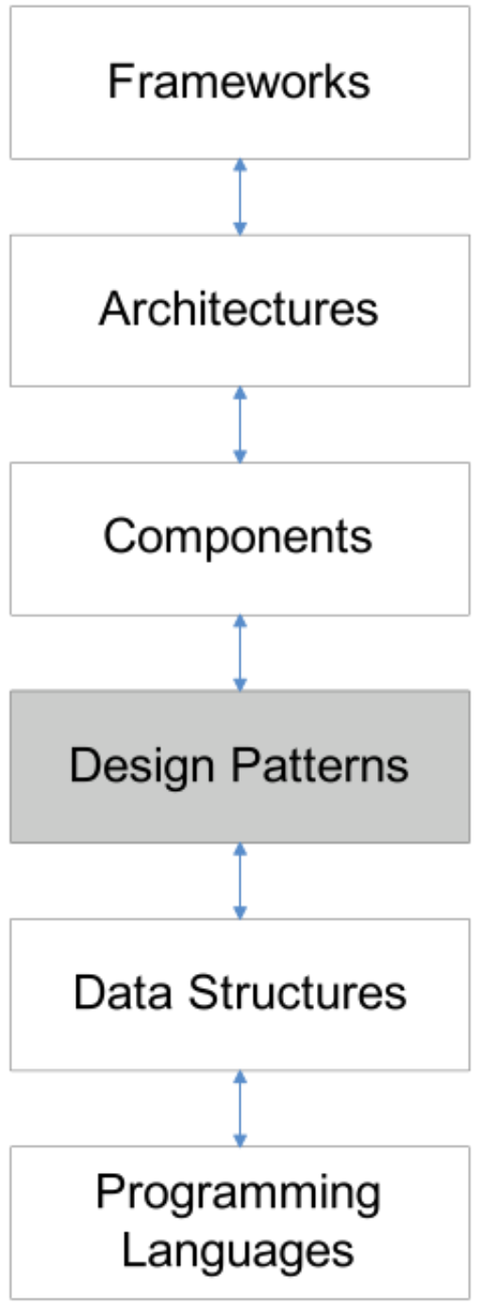
```
public class Contact
{
  private String name;

  public Contact(String name)
  {
    this.name = name;
  }

  public String toString()
  {
    return name;
  }
}
```

```
public class Main
{
  public static void main(String[] args)
  {
    List<Contact> contacts
          = new ArrayList<Contact>();

    contacts.add(new Contact("Mike"));
    contacts.add(new Contact("Mary"));
    contacts.add(new Contact("Jim"));

    System.out.println(contacts);
  }
}
```



Design Patterns

- A design pattern is a **proven** solution for a general design problem.
- It consists of communicating ‘objects’ that are customised to solve the problem in a particular context.
- Patterns have their origin in object-oriented programming; they are pre-packaged Object-oriented design knowledge that allows you to create more flexible and maintainable code.
- 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.



Why the need for Design Patterns?

- Change is a constant in software design e.g. bugs, new features, changes to design, new regulations, etc. All software changes, so your designs should be ready for it.

- They allow you to typically anticipate common ways that systems grow and change over time.

- The primary goal of any design pattern is to help you structure your code so it is flexible and resilient.

- All patterns let some part of the code vary independently of the other parts.

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.

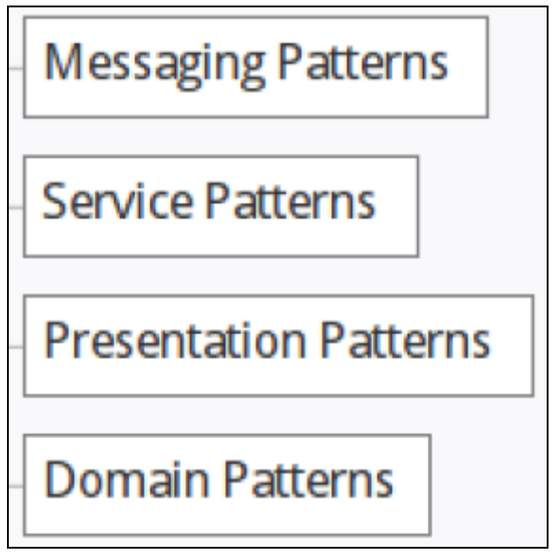
Design Patterns:

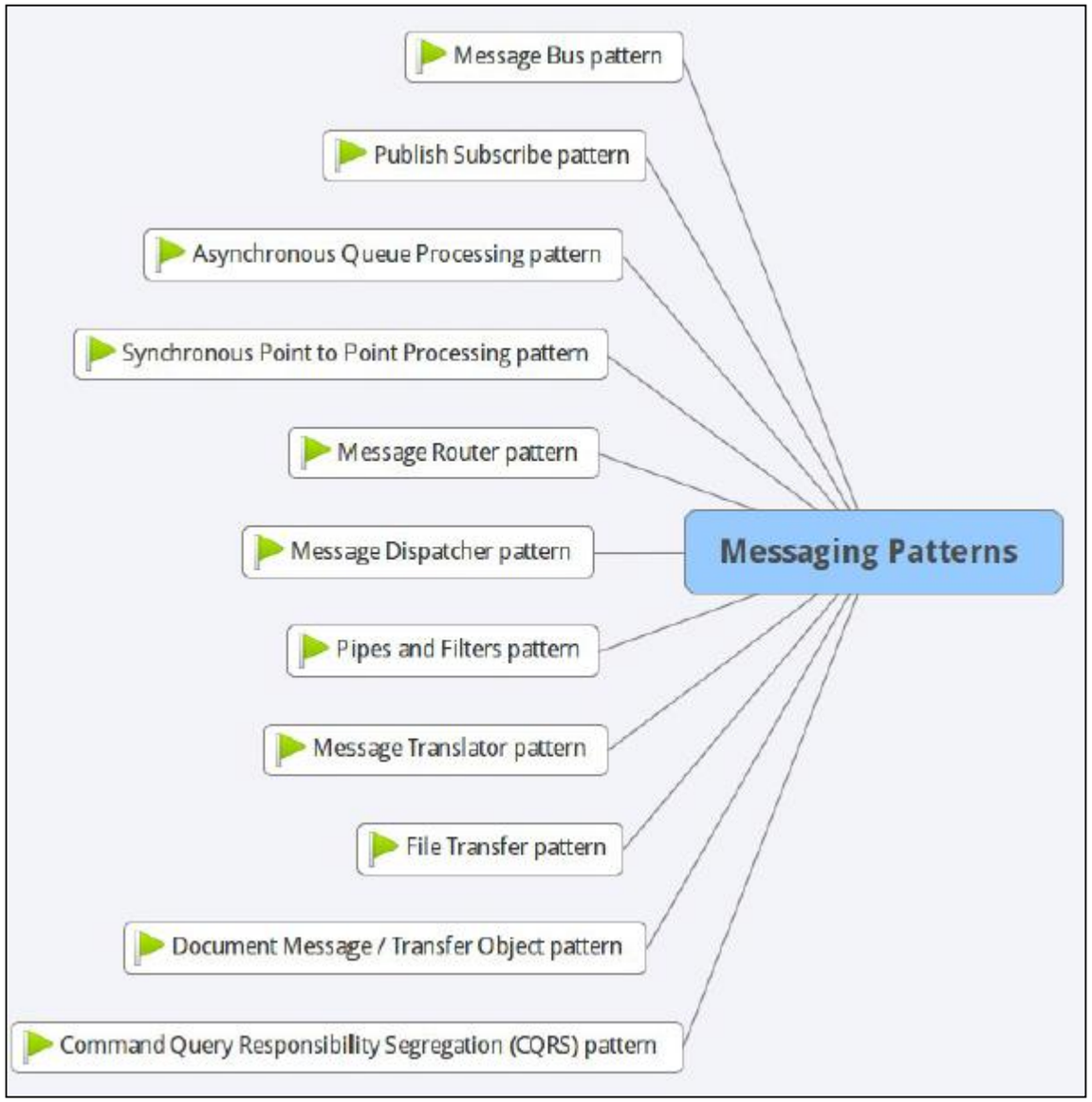
- Provides a scheme for refining the subsystems or components of a software system, or the relationships between them. It describes commonly recurring structure of communicating components that solves a general design problem within a particular context.

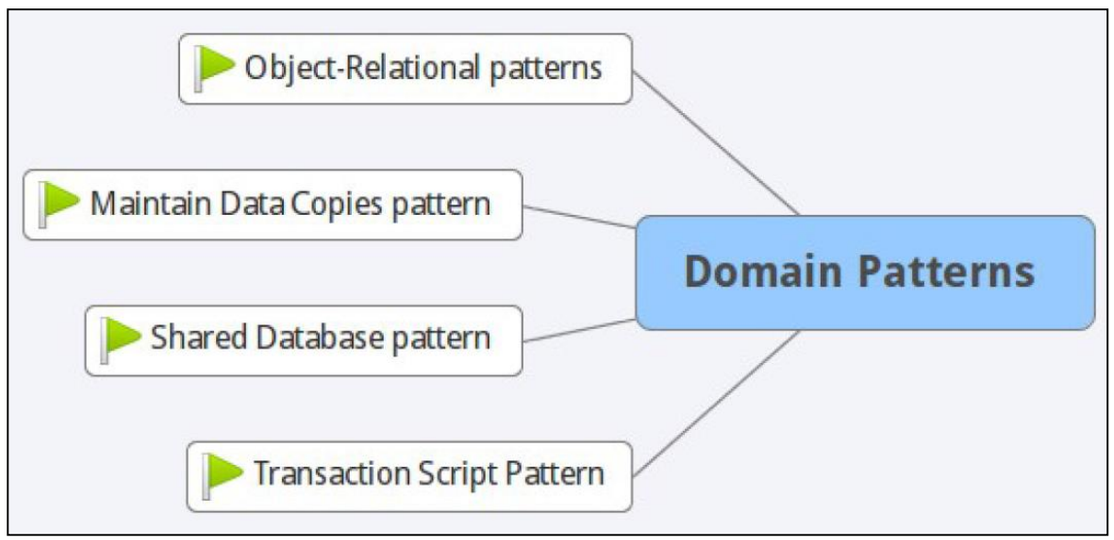
Idioms:

- 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.

Architectural Patterns Examples







Note: this area is covered in detail in the Design Patterns Module.

Singleton Pattern

```
public class FileLogger{

 private static FileLogger logger;

 private FileLogger(){
 }

 public static FileLogger getLogger(){
  if (logger == null){
    logger = new FileLogger();
  }
  return logger;
 }

 public boolean log(String msg){
  try{
    PrintWriter writer = new PrintWriter(new FileWriter("log.txt", true));
    writer.println(msg);
    writer.close();
  }
  catch (FileNotFoundException ex){
    return (false);
  }
  catch (IOException ex){
    return (false);
  }
  return (true);
 }
}
```

Design Patterns Examples

Creational Patterns: Used to construct objects such that they can be decoupled from their implementing system.

Structural Patterns: Used to form large object structures between many disparate objects.

Behavioral Patterns: Used to manage algorithms, relationships, and responsibilities between objects.

Object Scope: Deals with object relationships that can be changed at runtime.

Class Scope: Deals with class relationships that can be changed at compile time.

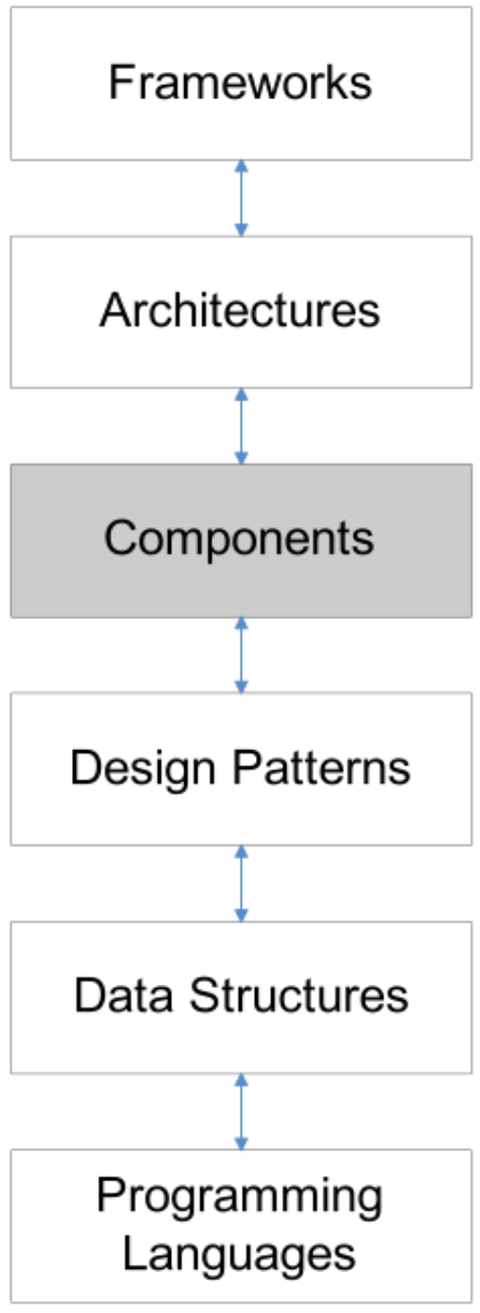
C	Abstract Factory	S	Decorator	C	Prototype
S	Adapter	S	Facade	S	Proxy
S	Bridge	C	Factory Method	B	Observer
C	Builder	S	Flyweight	C	Singleton
B	Chain of Responsibility	B	Interpreter	B	State
		B	Iterator	B	Strategy
B	Command	B	Mediator	B	Template Method
S	Composite	B	Memento	B	Visitor

Note: this area is covered in detail in the Design Patterns Module.

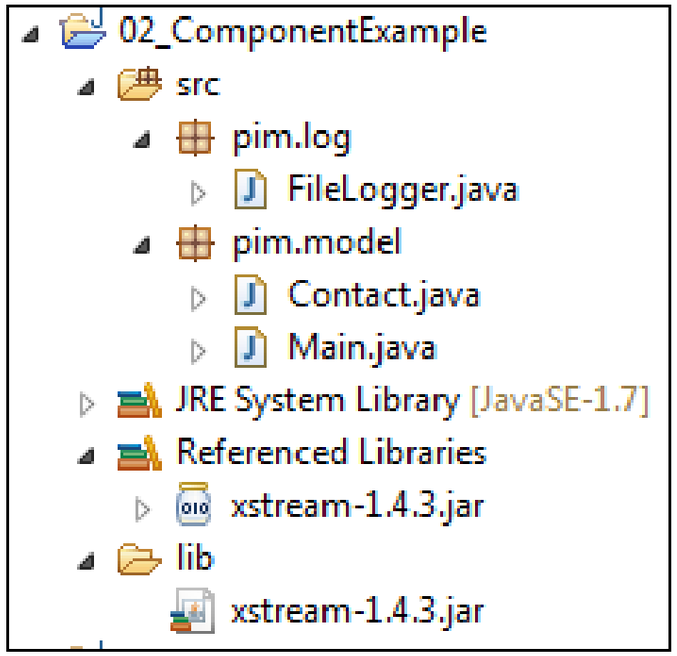
https://dzone.com/refcardz

Components

- Software components are binary units of:
  - independent production,
  - acquisition,
  - deployment
- that interact to form a functioning program. (Szyperski, 1998)
- Emphasis 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



Exploring a Component (xstream-1-4-3.jar)




```
<object-stream>
 <list>
  <pim.model.Contact>
   <name>Mike</name>
  </pim.model.Contact>
  <pim.model.Contact>
   <name>Mary</name>
  </pim.model.Contact>
  <pim.model.Contact>
   <name>Jim</name>
  </pim.model.Contact>
 </list>
</object-stream>
```

```
public class Main{
 public static void main(String[] args) throws IOException{
  FileLogger logger = FileLogger.getLogger();

  logger.log("Creating contact list");

  List<Contact> contacts = new ArrayList<Contact>();
  logger.log("Adding contacts");
  contacts.add(new Contact("Mike"));
  contacts.add(new Contact("Mary"));
  contacts.add(new Contact("Jim"));
  System.out.println(contacts);

  logger.log("Serializing contacts to XML");
  XStream xstream = new XStream(new DomDriver());
  ObjectOutputStream out =
          xstream.createObjectOutputStream
          (new FileWriter("contacts.xml"));
  out.writeObject(contacts);
  out.close();

  logger.log("Finished - shutting down");
 }
}
```

More Component Definitions

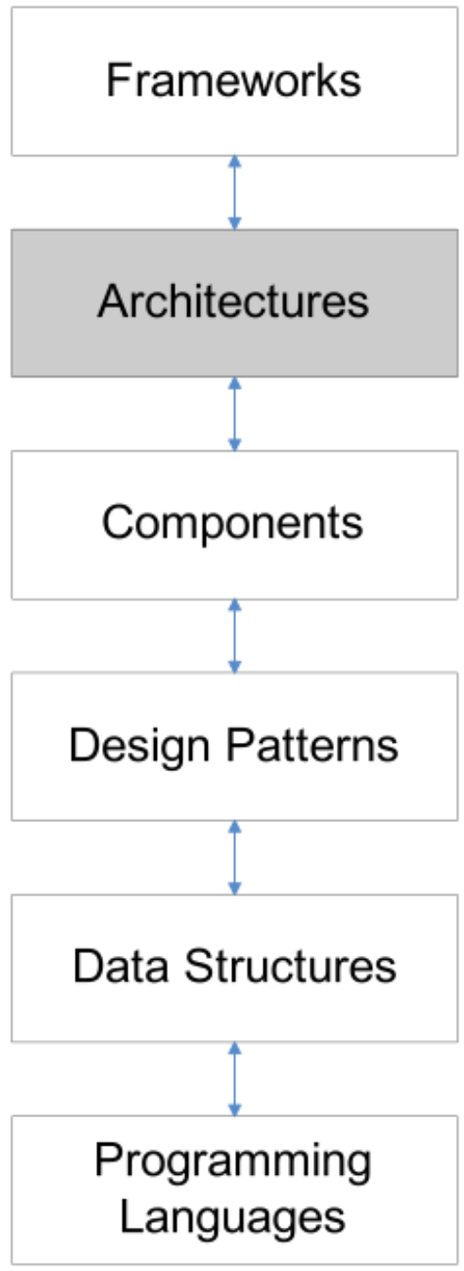
- "A component is a nontrivial, nearly independent, and replaceable part of a system that fulfills a clear function in the context of a well-defined architecture. A component conforms to and provides the physical realization of a set of interfaces." (Philippe Krutchen, Rational Software)

- "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)

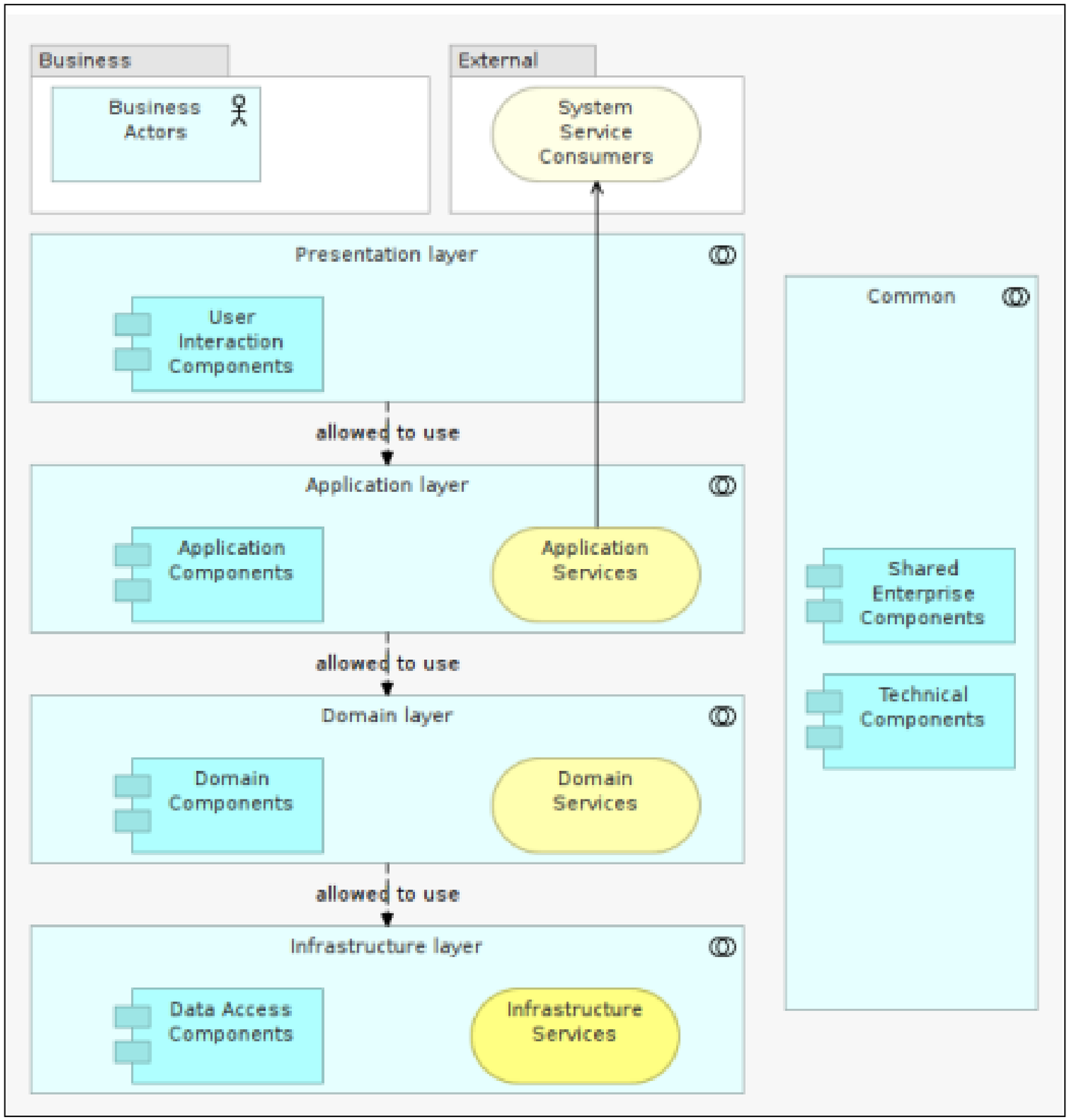
- "A component is a physical and replaceable part of a system that conforms to and provides the realization of a set of interfaces...typically represents the physical packaging of otherwise logical elements, such as classes, interfaces, and collaborations." (Grady Booch, Jim Rumbaugh, Ivar Jacobson, 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.

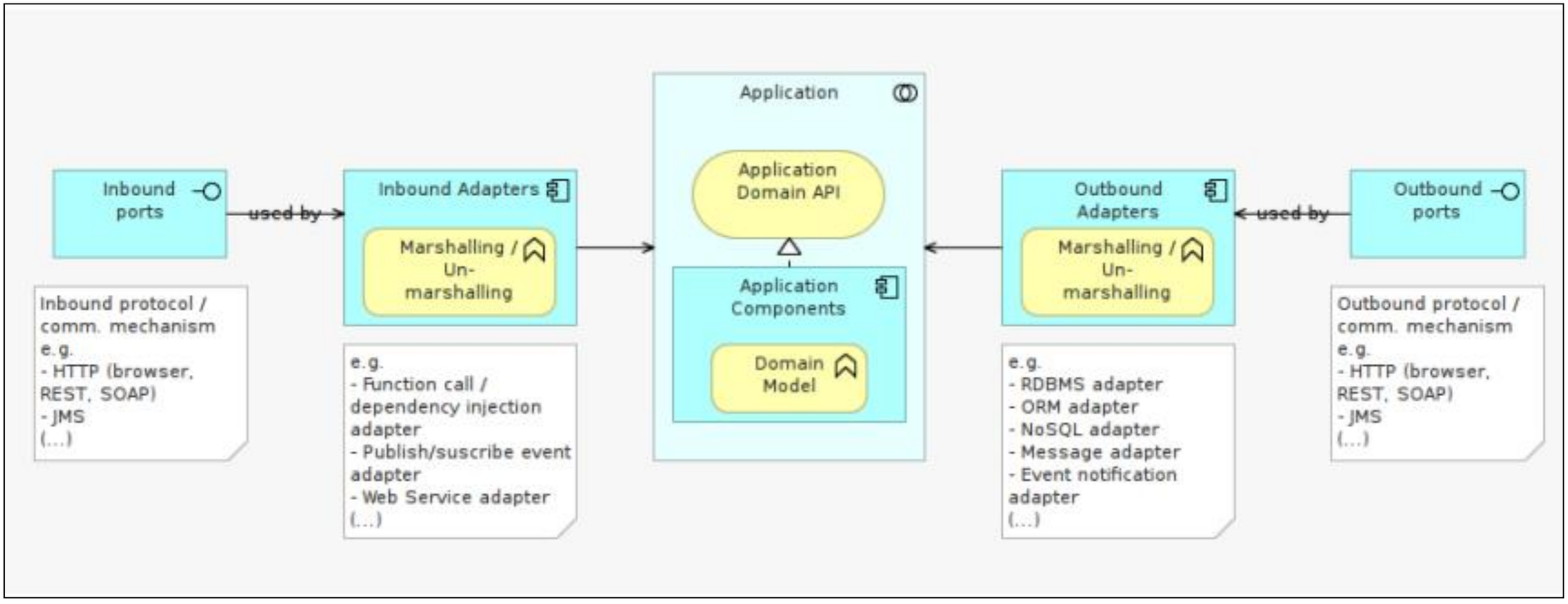


Architecture Example: Layered



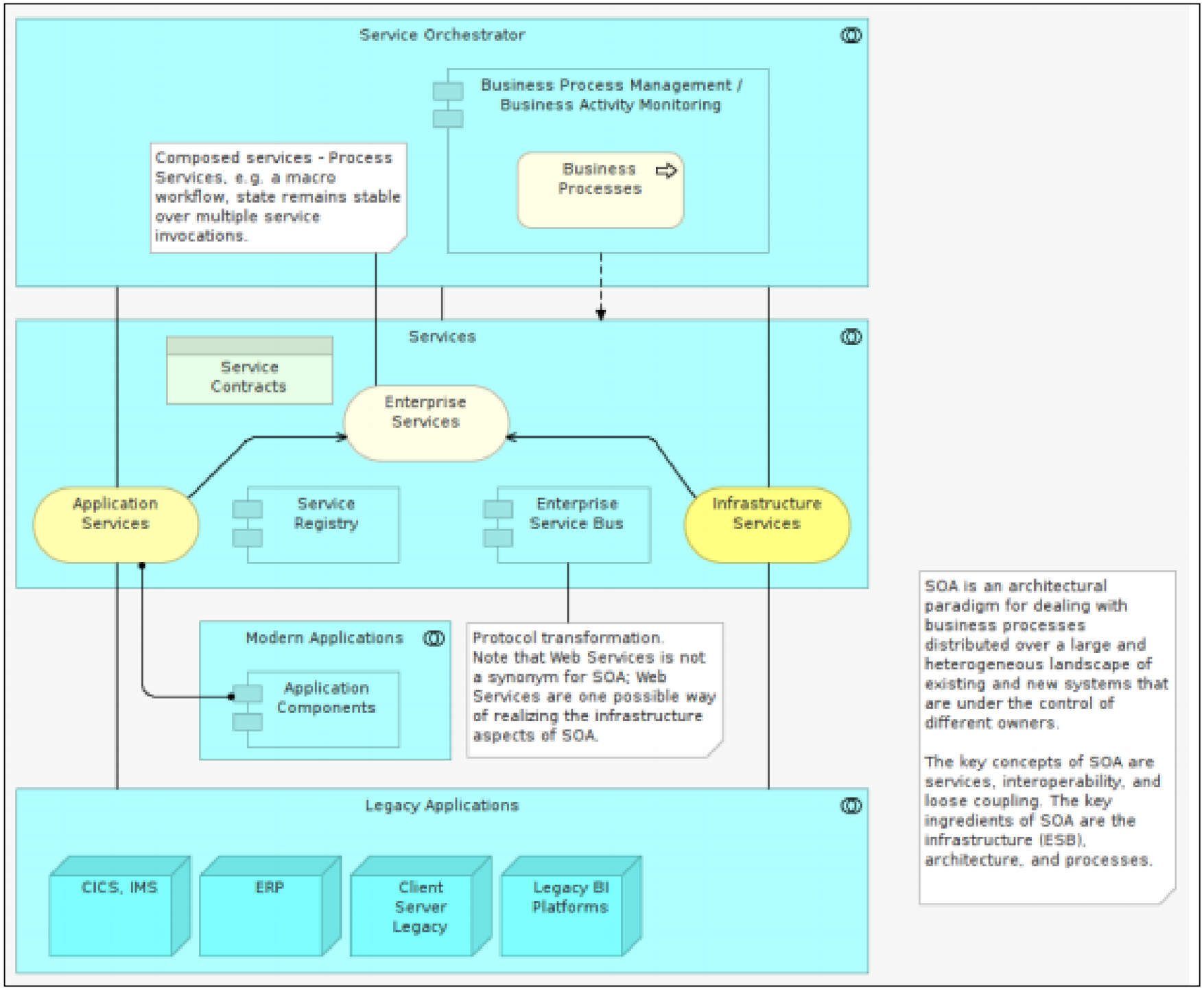
Source: Pierre Peclier, Design Patterns Module

Architecture Example: Ports and Adaptors



Source: Pierre Peclier, Design Patterns Module

Architecture Example: Service Oriented

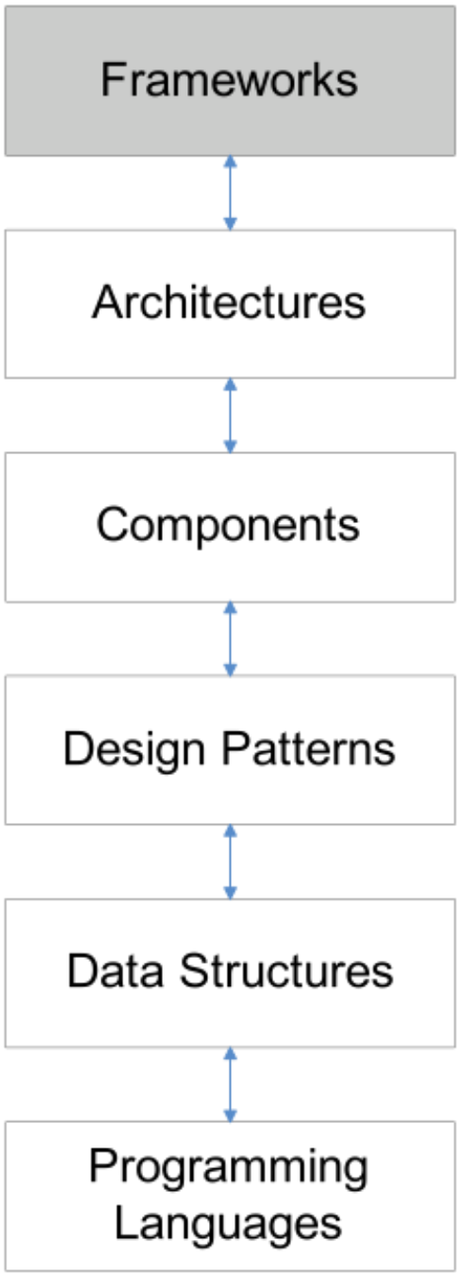


Source: Pierre Peclier, Design Patterns Module

Frameworks

- A framework is a set of related components which you:
  - Specialize
  - Integrate and/or
  - Instantiate
- to implement an application or subsystem.

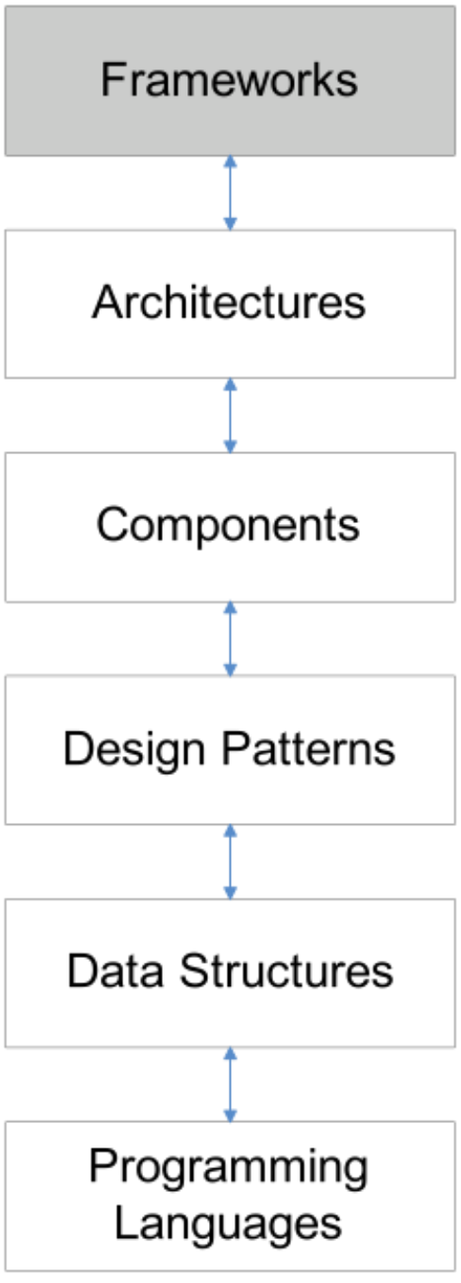
- A framework is usually a semi complete application containing dynamic and static components that can be customized to produce applications.



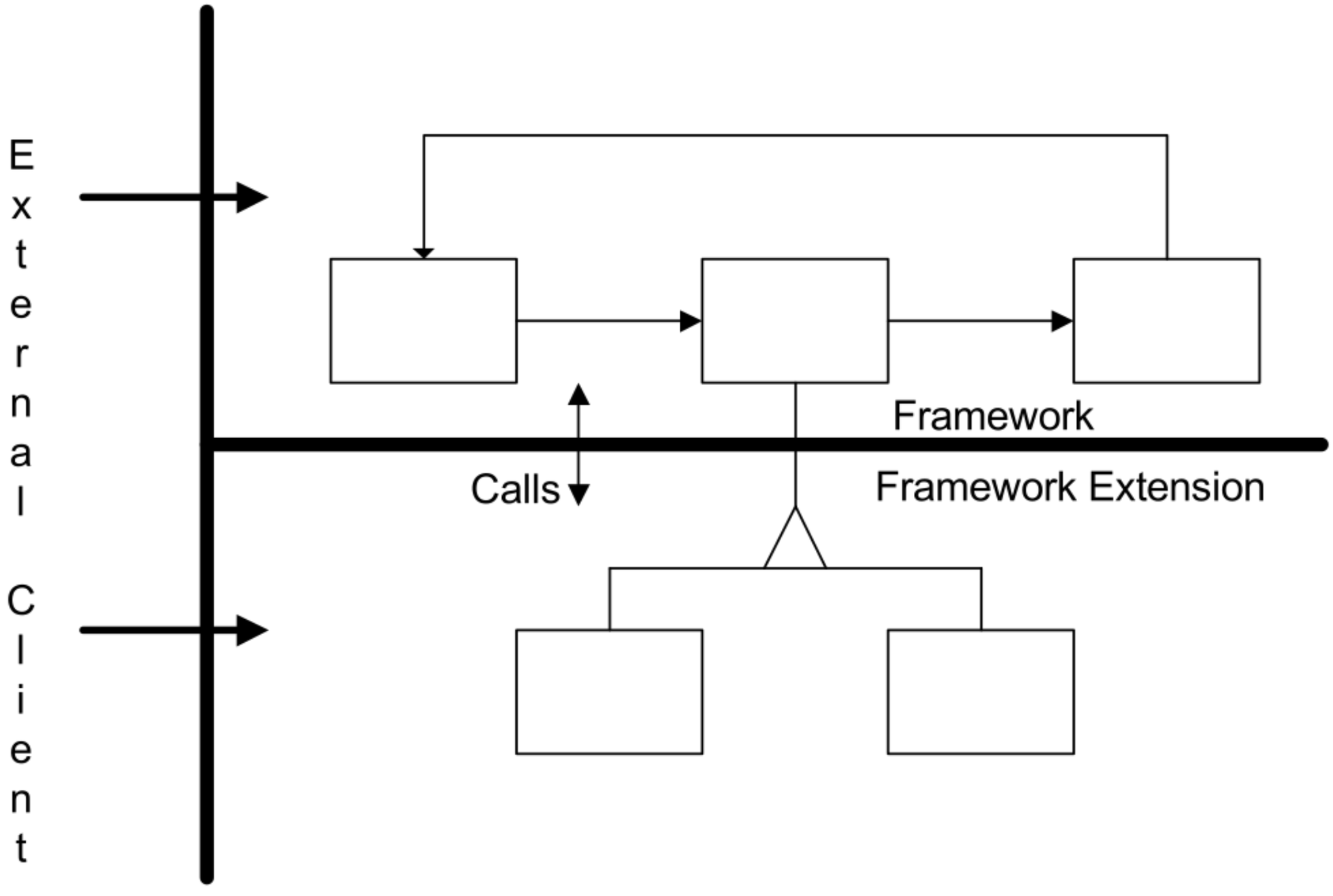
Frameworks

- 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)

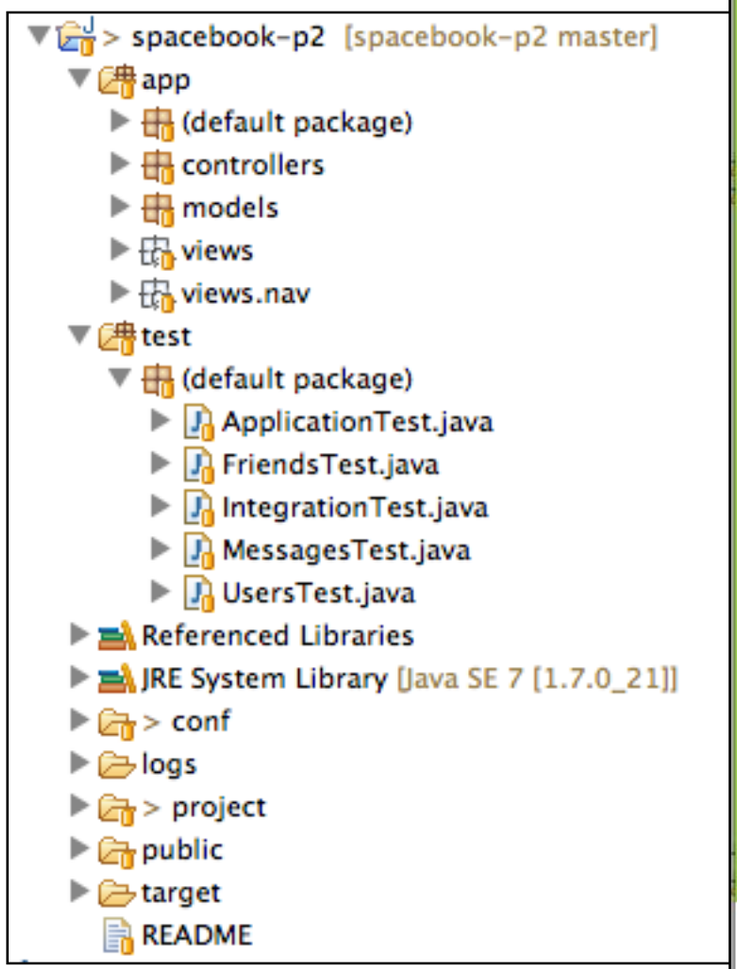
- Emphasize stable parts of the domain and their relationships and interactions.



Framework Structure



Framework Example



Agile Software Development Module

- Assumptions:
  - General Programming Ability (not necessarily java)
- Focus for this course:
  - SOLID Principles within OO Programming
  - Test Driven Software Development in Java
  - Effective Build Processes
  - Network Programming
  - Beyond Java (Kotlin)

Any questions?