Amrita Vishwa Vidyapeetham

TIFAC-CORE in Cyber Security

20CYS312 - Principles of Programming Languages
Assignment-01: Exploring Programming Paradigms

K Hitesh Manjunath

21st January, 2024

Paradigm 1: Procedural

Procedural programming is a programming paradigm that follows a linear and sequential approach to writing software. In procedural programming, a program is organized into procedures or routines that are executed one after another from the beginning to the end. These procedures are also known as functions, subroutines, or methods.In procedural programming data and functions are stored in separate memory location, while in OOP data and functions are stored in same memory location.

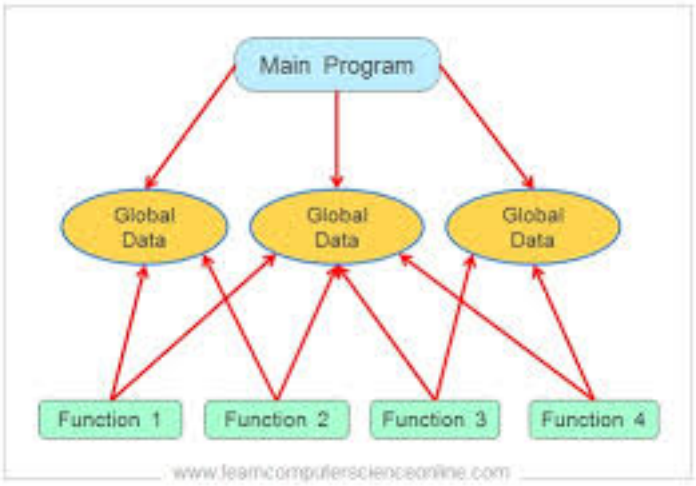

Figure 1: Procedural programming

Key Characteristics:

- Procedure-Centric:

The focus is on procedures or routines, which are blocks of code that perform specific tasks. These procedures are designed to be reusable and modular.

- Sequential Execution:

  Code is executed sequentially, and the flow of control moves from one statement to the next. This is often expressed using constructs like loops and conditionals.

- Limited Code Reusability:

  While procedures can be reused, procedural programming may lack the level of code reusability and abstraction found in other paradigms like object-oriented programming (OOP).

- No Data Hiding: Procedural programming does not emphasize data hiding or encapsulation. Variables are typically global or local to a procedure, and their values can be accessed directly.

Why Procedural Programming?
Procedural programming is valuable for its emphasis on clarity, readability, and structured development. Each procedure represents a specific task, promoting modularity and ease of debugging.The approach facilitates the step-by-step execution of procedures, aiding in problem identification.

Features

- The key features of procedural programming are given below:

  Predefined functions: A predefined function is typically an instruction identified by a name. Usually, the predefined functions are built into higher-level programming languages, but they are derived from the library or the registry, rather than the program. One example of a pre-defined function is 'charAt()', which searches for a character position in a string.

  Local Variable: A local variable is a variable that is declared in the main structure of a method and is limited to the local scope it is given. The local variable can only be used in the method it is defined in, and if it were to be used outside the defined method, the code will cease to work.

  Global Variable: A global variable is a variable which is declared outside every other function defined in the code. Due to this, global variables can be used in all functions, unlike a local variable.

  Modularity: Modularity is when two dissimilar systems have two different tasks at hand but are grouped together to conclude a larger task first. Every group of systems then would have its own tasks finished one after the other until all tasks are complete.

  Parameter Passing: Parameter Passing is a mechanism used to pass parameters to functions, subroutines or procedures. Parameter Passing can be done through 'pass by value', 'pass by reference', 'pass by result', 'pass by value-result' and 'pass by the name'.

Pros and Cons in Procedural
Pro's

- The ability to re-use the same code at different places in the program without copying it.
- An easier way to keep track of program flow.
- The ability to be strongly modular or structured.
- Needs only less memory.

Con's

- Data is exposed to whole program, so no security for data.
- Difficult to relate with real world objects.
- Difficult to create new data types reduces extensibility.

Example code in C++

Procedural v/s OOPs

Language for Paradigm 1: Ada

Ada is a statically typed, high-level programming language designed for reliability and safety in critical systems. It is heavily used in embedded real-time systems.
Reasons to use Ada:

- Ease of learning
- Wide range of target processors supported.
- Rich and mature development environments and tool sets.
- It includes control structures like 'if,' 'case,' 'loop,' and 'while,' providing tools for structured flow control in procedural programming.
- Ensures that variables have well-defined types.

There exists another language close and interoperable with Ada, it is "SPARK". SPARK is a subset of Ada, designed so that the written code is ameneable to automatic proof. It provides a level of assurance with regard to the correctness of your code that is much higher than regular programming language.

It really shines in low-level applications:

- Direct interfacing with hardware.
- Soft or hard real-time systems.
- Low-level systems programming.
- Embedded systems with low memory requirements.

Procedural Oriented Programming	Object-Oriented Programming
In procedural programming, the program is divided into small parts called ***functions***.	In object-oriented programming, the program is divided into small parts called ***objects***.
Procedural programming follows a ***top-down approach***.	Object-oriented programming follows a ***bottom-up approach***.
There is no access specifier in procedural programming.	Object-oriented programming has access specifiers like private, public, protected, etc.
Adding new data and functions is not easy.	Adding new data and function is easy.
Procedural programming does not have any proper way of hiding data so it is ***less secure***.	Object-oriented programming provides data hiding so it is ***more secure***.
In procedural programming, overloading is not possible.	Overloading is possible in object-oriented programming.
In procedural programming, there is no concept of data hiding and inheritance.	In object-oriented programming, the concept of data hiding and inheritance is used.
In procedural programming, the function is more important than the data.	In object-oriented programming, data is more important than function.
Procedural programming is based on the ***unreal world***.	Object-oriented programming is based on the ***real world***.
Procedural programming is used for designing medium-sized programs.	Object-oriented programming is used for designing large and complex programs.
Procedural programming uses the concept of procedure abstraction.	Object-oriented programming uses the concept of data abstraction.
Code reusability absent in procedural programming,	Code reusability present in object-oriented programming.
Examples: C, FORTRAN, Pascal, Basic, etc.	**Examples:** C++, Java, Python, C#, etc.

Figure 2: Differences between OOPs and Procedural

Fundamentals in Ada

- Ada allows the definition of procedures and functions, enabling developers to encapsulate sets of instructions into reusable and modular units.
- Ada's tasking model supports concurrent programming, allowing developers to create parallel and distributed systems.
- There is mostly no type reference.

How Ada can be useful in real-world applications?
Ada ensures the development of robust and dependable systems. Its static verification tools enhance code correctness, while exception handling allows graceful error management.
Ada's high-level abstractions, portability across platforms, and support for formal methods contribute to code readability, maintainability, and the formal verification of critical systems.

Real-world applications in Ada

- Aerospace and Defense Systems
- Railway Systems
- Medical Devices
- Energy Sector

Basic Program in Ada

```
with Ada.Text_IO;

procedure Greet is
begin
   --  Print "Hello, World!" to the screen
   Ada.Text_IO.Put_Line ("Hello, World!");
end Greet;
```

Figure 3: Hello world program

- "with" - Works like import function in python, it imports all the standard library module Ada, which contains Ada.TextIO package, which contains a procedure to print text.
- Greet is a procedure, and the main entry point for our program.
- 'begin' and 'end' define the beginning and end of the code.
- 'Putline' is a inbuilt function in the module Ada.TextIO which prints.

Program with IF/THEN/ELSE

```
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;

procedure Check_Positive is
 N : Integer;
 begin
 -- Put a String
 Put ("Enter an integer value: ");

 -- Read in an integer value
 Get (N);

 if N > 0 then
 -- Put an Integer
 Put (N);
 Put_Line (" is a positive number");
 end if;
end Check_Positive;
```

Figure 4: Positive/Negative

```
Enter an integer value: 7
7 is a positive number
```

Figure 5: Output

- This example declares an integer variable N, prompts the user for an integer, checks if the value is positive and, if so, displays the integer's value followed by the string " is a positive number". If the value is not positive, the procedure does not display any output.
- This example illustrates some of the basic functionality for integer input-output. The relavant subprograms are in the predefined package Ada.IntegerTextIO and include the Get procedure (which reads in a number from the keyboard) and the Put procedure (which displays an integer value).
- **Loops**
- I is local to the loop, so you cannot refer to I outside the loop. Although the value of I is incremented at each iteration, from the program's perspective it is constant. An attempt to modify its value is illegal; the compiler would reject the program.
- Integer'Image is a function that takes an Integer and converts it to a String. It is an example of a language construct known as an attribute, indicated by the ' syntax, which will be covered in more detail later.
- The end loop marks the end of the loop

```
with Ada.Text_IO; use Ada.Text_IO;

procedure Greet_5a is
begin
 for I in 1 .. 5 loop
 -- Put_Line is a procedure call
 Put_Line ("Hello, World!"
 & Integer'Image (I));

 end loop;
end Greet_5a;
```

Figure 6: For-loop

```
Hello, World! 1
Hello, World! 2
Hello, World! 3
Hello, World! 4
Hello, World! 5
```

Figure 7: Output

- The "step" of the loop is limited to 1 (forward direction) and -1 (backward).

Declarative regions

- Ada draws a clear syntactic separation between declarations, which introduce names for entities that will be used in the program, and statements, which perform the processing. The areas in the program where declarations may appear are known as declarative regions.
- In any subprogram, the section between the is and the begin is a declarative region. You can have variables, constants, types, inner subprograms, and other entities there.
- We've briefly mentioned variable declarations in previous subsection. Let's look at a simple example, where we declare an integer variable X in the declarative region and perform an initialization and an addition on it: Example in page 8.

Procedural programming in Ada

- Procedural programming in Ada involves structuring code around procedures and functions, encapsulating related functionality into packages, and emphasizing modularity and sequential execution. It provides a straightforward approach to solving problems by breaking them down into smaller, more manageable units.

Example Code: Code in page no.9

- **Explanation:**
- The provided Ada program exemplifies procedural programming by calculating the factorial of a user-input positive integer. It imports the Ada.TextIO package for handling

```ada
with Ada.Text_IO; use Ada.Text_IO;

procedure Main is
X : Integer;
begin
X := 0;
Put_Line ("The initial value of X is "
& Integer'Image (X));

Put_Line ("Performing operation on X...");
X := X + 1;

Put_Line ("The value of X now is "
& Integer'Image (X));
end Main;
```

Figure 8: Declarative

```
The initial value of X is 0
Performing operation on X...
The value of X now is 1
```

Figure 9: Output

input and output. The recursive Factorial function determines the factorial of a given number, while the Main procedure prompts the user for input, checks for non-negativity, and displays the calculated factorial.
The program demonstrates a modular and structured approach, adhering to procedural programming principles. Its execution begins with the Main procedure, showcasing the procedural paradigm's emphasis on organized code and step-by-step execution.

Key Feautures of ada that support Procedural programming

- Ada supports DbC(Design By Contract), which is a programming technique that emphasizes the use of preconditions and postconditions to ensure that procedures and functions are used correctly. This helps catch errors at compile-time and improves the reliability of the code.
- Provides built-in constructs for defining and calling subroutines, allowing developers to break down complex tasks into smaller, more manageable pieces. Procedures and functions can be called multiple times within a program, allowing for code reuse and modularity.

Case Study: Space Missions

- **Navigation and Control Systems:**

  Ada is used in the development of navigation and control systems for spacecraft. Procedural programming in Ada helps implement precise algorithms for trajectory calculations, attitude control, and orbit adjustments.

  Communication Protocols:

```ada
with Ada.Text_IO;

procedure Factorial_Program is

   function Factorial(N : Positive) return Positive is
   begin
      if N = 0 then
         return 1;
      else
         return N * Factorial(N - 1);
      end if;
   end Factorial;

   -- Main procedure
   procedure Main is
      Num : Positive;
   begin
      Ada.Text_IO.Put("Enter a positive integer: ");
      Ada.Text_IO.Get(Item => Num);

      if Num >= 0 then
         Ada.Text_IO.Put_Line("Factorial of " & Positive'Image(Num) & " is: " & Positive'Image(Factorial(Num)));
      else
         Ada.Text_IO.Put_Line("Please enter a non-negative integer.");
      end if;
   end Main;
begin

   Main;
end Factorial_Program;
```

Figure 10: Ada code with procedural programming

```
Enter a positive integer: 5
Factorial of 5 is: 120
```

Figure 11: Output

Spacecraft communicate with Earth-based stations, satellites, and other spacecraft.

Ada's support for low-level programming and communication protocols is valuable in ensuring reliable data transfer.

Onboard Systems:
Ada is employed for developing software that controls onboard systems and instruments. Procedural programming is used to implement routines for scientific instruments, data processing, and telemetry.

Multi-Language Integration:
Ada's ability to integrate with other programming languages is valuable when working with legacy systems or incorporating components developed in different languages.

Fault Detection and Recovery:
Ada's exception handling capabilities play a crucial role in detecting and recovering from faults. Procedural programming in Ada helps design fault-tolerant systems that can continue operating despite unexpected events.

Paradigm 2: Event-Driven

The concept of event-driven has been a fundamental part of programming for many decades and has evolved over time with advances in technology and programming languages. Event-driven programming is a paradigm where the flow of a program is determined by events such as user actions, sensor outputs, or messages from other programs or threads.It is commonly used in graphical user interfaces, web development, and systems where asynchronous input/output operations are prevalent. Event-driven programming enables responsiveness and interactivity by allowing the program to efficiently handle a wide range of events without blocking the overall execution.

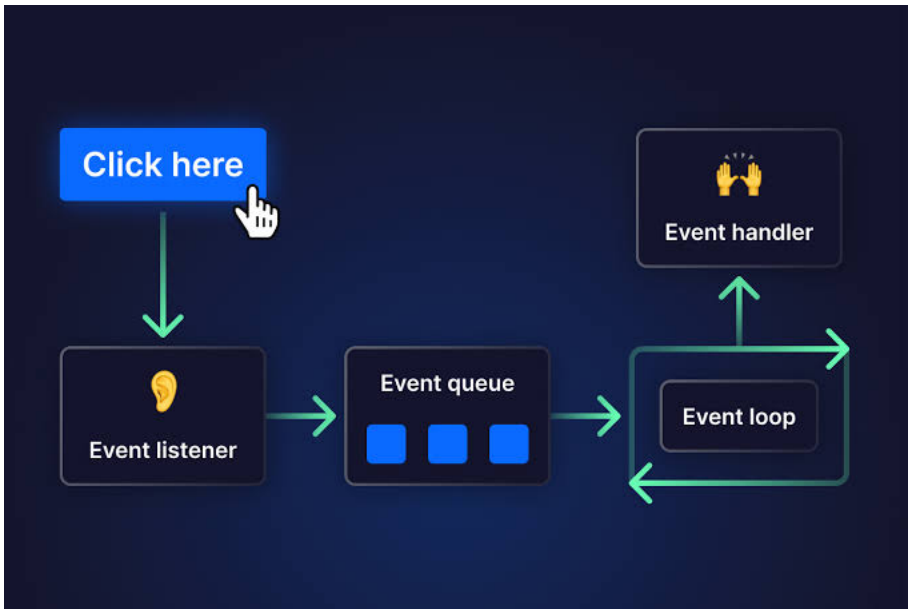

Figure 12: Event-driven Architecture

Characteristics of Event-Driven programming

- **Asynchronous Execution:**Events in an event-driven system can occur independently of the main program flow, leading to asynchronous execution. The program responds to events as they happen, without waiting for each operation to complete before moving on.
- **Event Handlers**:It relies on the use of event handlers or callbacks. These are functions or procedures that are executed in response to specific events. They define how the program should react when a particular event occurs.
- **Event-Triggered Flow:**The flow of the program is determined by events rather than a predefined sequence of instructions. Events act as triggers for the execution of associated event handlers, allowing for a dynamic and responsive program behavior.
- **Message Passing**:Communication between components often involves message passing through events. Components can send messages to notify others about specific actions or changes, facilitating inter-component communication.

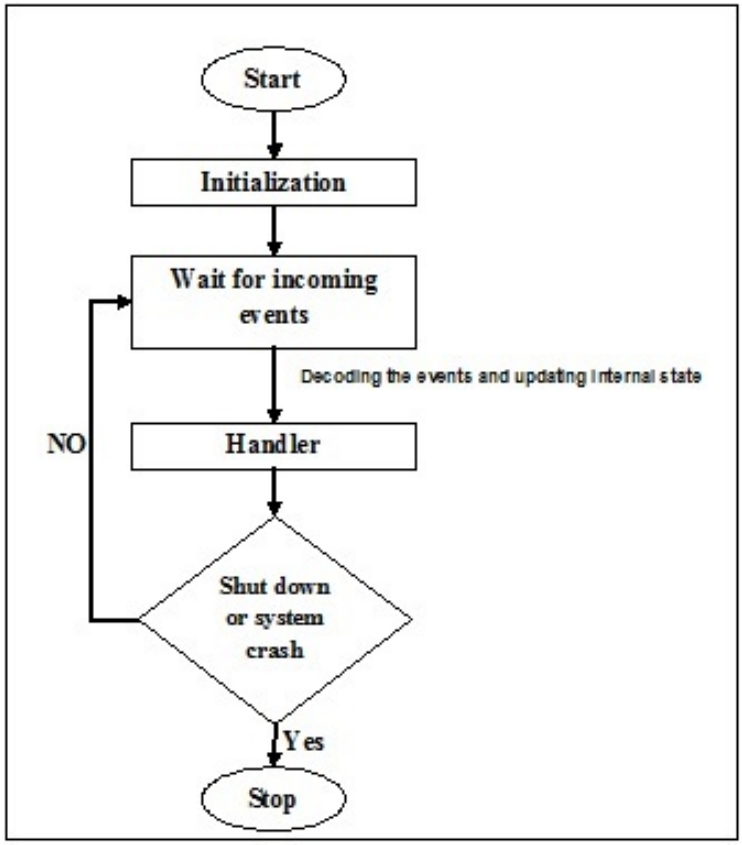

Figure 13: Flowchart

- **Flexibility and Scalability:**Event-driven systems are flexible and scalable, making it easier to extend and modify the functionality without affecting the entire program. New events and event handlers can be added without significant changes to the existing codebase.

Pros and Cons of Event-Driven Programming Pro's

- Responsive Applications: Event Driven Programming allows applications to effectively respond to user input, resulting in a more dynamic and user-friendly experience. The event loop and event queue maintain the timely processing of events, ensuring that user interactions are effectively handled.

  Concurrency: Asynchronous event handling enables applications to execute multiple tasks concurrently. This capability can improve the overall performance and responsiveness of an application, particularly in situations where tasks are resource-intensive or time-consuming.

  Modularity and Maintainability: The separation of concerns in event-driven applications, through distinct event handlers and event management, promotes modularity and maintainability. Developers can focus on individual event handlers, making it simpler to comprehend, adjust, and augment the software.

Scalability: Asynchronous event-driven architecture allows applications to efficiently utilize system resources, making it possible to scale both vertically and horizontally.

Real-Time Processing: In the context of real-time applications, Event Driven Programming enables the processing of events as they occur, ensuring the continuous distribution of up-to-date information and consistent system responsiveness.

Wide Range of Applications: Event Driven Programming can be applied across diverse domains, including web applications, graphical user interfaces, server-side systems, and data-driven applications.

Con's

- **Complexity:** The asynchronous nature of event-driven applications can increase the software's complexity. Ensuring correct synchronization, managing race conditions, and addressing deadlock scenarios may necessitate extensive effort and precision in coding.

  Debugging Difficulties: Debugging event-driven applications can prove challenging, especially when dealing with concurrency, as the order of event execution is not predetermined and may vary during runtime. This unpredictability can complicate the process of identifying and resolving issues.

  Event Handling Overhead: The execution of event handlers and management of events demand additional system resources. Furthermore, the event loop and event queue require constant monitoring, potentially impacting performance.

  Steep Learning Curve: Developers who are unfamiliar with Event Driven Programming may experience a steep learning curve, particularly when grappling with complex concurrency behaviours and synchronisation.

  Dependencies on External Libraries: In certain programming languages and environments, Event Driven Programming might rely on external libraries for managing events and handling asynchronous tasks. This dependence on external code may complicate deployment and maintenance.

Features

- **Modularity and Reusability** Modularity and Reusability are two key features of Event Driven Programming that make it simple to construct, maintain, and enhance applications. In this paradigm, event handlers and other components are separated, allowing developers to compose their applications using well-defined modules. This separation between responsibility and implementation greatly reduces code duplication, as each module is only responsible for a specific task.
  Developing modular code can result in numerous benefits, such as:

  –

  1.Improved code readability and maintainability

2. Minimised integration efforts

3. Enhanced testing and debugging potential

4. Reduced development time

5. Ability to reuse components across multiple projects

Reusability is another significant advantage of modular code in Event Driven Programming. Event handlers, utility functions, and other application components can be easily repurposed for various projects, simplifying the development process and resulting in a more consistent codebase. Reusability also boosts the efficiency of collaboration amongst developers, as a shared codebase allows them to quickly understand and analyse the source code.

Example: Imagine a software application with a menu system containing multiple buttons. Each button has a distinct action associated with it. With modularity and reusability, you can create multiple modules that handle the actions corresponding to the specific buttons instead of writing repetitive code in one large file.

Flexibility and Scalability Flexibility and Scalability are two indispensable features of Event Driven Programming that allow applications to meet changing demands and grow over time.

- Flexibility implies that an application can modify its behaviour or extend its functionality without requiring substantial code alterations. Scalability refers to an application's ability to maintain optimal performance as the workload increases or resources are added to the system.

  Event Driven Programming fosters flexibility by employing loosely-coupled components and promoting modularity. This architecture ensures that modifying, extending, or replacing a single event handler does not impact other parts of the application.

  Thus, developers can readily adapt their software to new requirements, emerging technologies, or changing business environments.

  Some of the advantages of flexibility in Event Driven Programming include:

  Efficient adaptation to changing requirements or users' needs Swift incorporation of new functionality or third-party services Opportunities for continuous software improvement Decreased risk of system obsolescence Scalability is crucial for applications that must accommodate growing workloads or operate with variable resources.

  Event Driven Programming, particularly through its support of asynchronous and concurrent execution, enables software to efficiently utilise available resources, thereby maintaining consistent performance as system demands evolve.

  Some of the advantages of scalability in Event Driven Programming include:

  Efficient resource utilisation

Performance consistency under heavy workloads

Effective horizontal and vertical scaling

Ability to adapt to diverse deployment environments

Language for Paradigm 2: Vue.js

VueJS is an open source progressive JavaScript framework used to develop interactive web interfaces. It is one of the famous frameworks used to simplify web development. VueJS focusses on the view layer. It can be easily integrated into big projects for front-end development without any issues. **Features**

- **Data Binding** The data binding feature helps manipulate or assign values to HTML attributes, change the style, assign classes with the help of binding directive called v-bind available with VueJS.
- **Components** Components are one of the important features of VueJS that helps create custom elements, which can be reused in HTML.
- **Directives** VueJS has built-in directives such as v-if, v-else, v-show, v-on, v-bind, and v-model, which are used to perform various actions on the frontend.
- **Computed Properties** This is one of the important features of VueJS. It helps to listen to the changes made to the UI elements and performs the necessary calculations. There is no need of additional coding for this.
- **Animation/Transition** VueJS provides various ways to apply transition to HTML elements when they are added/updated or removed from the DOM. VueJS has a built-in transition component that.
- **Watchers** Watchers are applied to data that changes. For example, form input elements. Here, we don't have to add any additional events. Watcher takes care of handling any data changes making the code simple and fast.

Comparision with other frameworks

VueJS v/s React Virtual DOM Virtual DOM is a virtual representation of the DOM tree. With virtual DOM, a JavaScript object is created which is the same as the real DOM. Any time a change needs to be made to the DOM, a new JavaScript object is created and the changes are made.Later, both the JavaScript objects are compared and the final changes are updated in the real DOM. VueJS and React both use virtual DOM, which makes it faster.

- **v/s Angular**
  - **Similarities** VueJS has a lot of similarities with Angular. Directives such as v-if, v-for are almost similar to ngIf, ngFor of Angular.
  - They both have a command line interface for project installation and to build it. VueJS uses Vue-cli and Angular uses angular-cli. Both offer two-way data binding, server side rendering, etc.

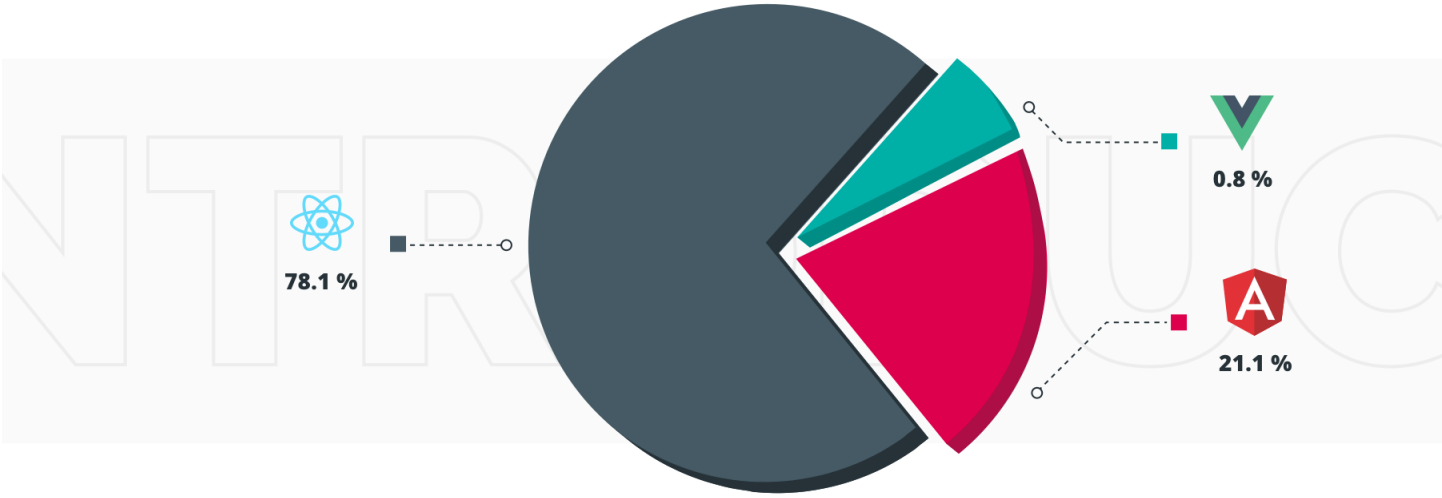

Figure 14: Ratio of Angular, Vue, React

- **Typescript** Angular uses TypeScript for its coding. Users need to have knowledge of Typescript to get started with Angular.
- However, we can start with VueJS coding anywhere in jsfiddle or codepen using the cdn library. We can work with standard JavaScript, which is very easy to start with.

v/s Polymer

- Polymer library has been developed by Google. It is used in many Google projects such as Google I/O, Google Earth, Google Play Music, etc. It offers data binding and computed properties similar to VueJS.

Event-Driven Programming in VueJS

- 1. Data Binding: - Vue.js supports two-way data binding, allowing changes in the user interface to automatically update the underlying data and vice versa. - Directives like v-model enable the binding of form inputs to data properties, creating a seamless connection between the UI and data.

  2. Reactivity System: - Vue.js uses a reactivity system to track changes in data. When data properties are modified, Vue automatically updates the associated parts of the virtual DOM and efficiently applies the changes to the real DOM. - This reactivity system is the foundation for Vue's responsiveness to events and data changes.

  3. Event Handling: - Components in Vue.js can emit and listen for events. Custom events can be defined and emitted by child components and then captured and handled by parent components. - The @ symbol in Vue templates is commonly used for event handling. For example, @click is used to handle the click event.

- **Drawbacks of Event-Driven programming in VueJS**
  Event-driven programming might become complex, Managing and synchronising states across the system is powerful but also more challenging and complex.
  It might be prone to errors Since we rely on events (typically described by their name), we might stumble upon collisions or errors. There's a massive difference between 'click' and 'clicked,' even though they feel the same. One will work, and the other – will silently do nothing.
  Calling non-existing methods from the parent will trigger an error, but events don't require listeners (or handlers). They happen and do not know if anyone would care to act.
  The flow of actions is not centralised, and it's like a flowing river. Events can occur everywhere, and handlers might be spread as well.

Case study: Fathom

- Fathom is a financial intelligence platform designed to integrate seamlessly with your existing accounting product and provide you with insightful analysis and reports. Fathom is a financial analysis tool launched in 2015 by a team from Brisbane, Australia. Using Aurelia and Vue.js, they deliver a useful tool for performing in-depth financial analyses and reporting to 20,000 businesses around the world.

  The product struggled with scaling, inconsistent architecture, and duplicate code which was difficult to debug and maintain. There was another primary framework required [VueJS]. Vue's flexibility played an important role. Although Vue abounds in multiple features, there were couple of drawbacks.

- **Challenges**
  Difficulties with scaling due to code bundling.
  Outdated product design with fragmented styles.
  Duplicate code and inconsistent architecture.
  Difficult to debug and maintain.
  Use of old or poorly supported libraries causing a significant business risk.

- **Solutions**
  Incrementally transitioning legacy features across and writing new features into new architecture.
  Creating a proof of concept.
  Putting a demo together and presenting it to management.

- **Result**
  Codebase easy to maintain,test, and share across multiple applications.
  The stylesheet cut down by 67.3
  The app's memory usage reduced by half.

Analysis

Procedural Programming in Ada:

- Strengths:
  Ada is designed with a strong focus on safety and reliability, making it suitable for critical systems where errors can have severe consequences.

  It tasking model allows for efficient and controlled concurrency, making it well-suited for applications that require parallel processing.

  It promotes clear and readable code through its strong typing and structured programming constructs.

  Weaknesses:
  Ada may have a steeper learning curve for developers unfamiliar with its syntax and features.

  Notable Features:
  It's tasking model supports concurrent programming, enabling developers to manage parallel execution effectively.

  Ada's strong typing helps catch errors at compile-time, enhancing code reliability.

Event-Driven Programming in Vue.js:

- **Strengths:**
  Vue.js is known for its simplicity and ease of integration, making it accessible for developers with varying levels of experience.

  Vue.js allows for a more flexible and dynamic development process, adapting well to changing requirements.

- **Weaknesses:** highly demanding or computation-intensive applications, Vue.js may face performance challenges compared to lower-level languages.

  Integrating Vue.js into large and complex projects might require careful planning and management.

  Notable Features:
  Vue.js promotes a component-based architecture, facilitating the development of modular and reusable UI elements.

  The Vue CLI offers powerful tools for project scaffolding, development, and testing.

Comparison

Ada, a general-purpose programming language, primarily supports procedural programming, which involves organizing code into procedures or routines. On the other hand, Vue.js is a JavaScript framework that follows an event-driven programming paradigm, where actions or events trigger corresponding responses.

Ada programs are structured around procedures and functions, emphasizing step-by-step

execution. It follows a linear control flow, with procedures being called sequentially.

Ada relies on the concept of variables and data structures to manage program state.

Ada supports concurrent programming through tasks and is known for its strong support for parallelism.

VueJS

Vue.js organizes code around components and relies on events to trigger actions.

It follows a non-linear control flow, where components react to events asynchronously.

Vue.js manages state using a reactive data model and employs a virtual DOM for efficient updates.

Vue.js supports concurrency through its reactive model, allowing multiple components to update independently.

Similarities
Both of them encourage modular design,promoting code organization and reusability. Support abstraction to Simplify complex systems.Provides mechanisms for handling concurrency, allowing for parallel execution.

Differences

- Ada - linear control flow, Vue.js - Non-linear control flow.
- Ada manages state through variables and data structures, Vue.js uses reactive data model by automatically updating the DOM based on changes in component state.
- Ada is used in safety-critical systems whereas VueJS is used in web Applications and user interfaces.

Challenges Faced

The First challenge I faced was Finding proper information regarding paradigms as I was confused between some. Even though Ada is a powerful language finding articles based on procedural paradigm was very hard. Ada is basically used in aerospace or defenses. General resources might not be sufficient for covering all aspects in procedural programming.In event-driven paradigm,It's basically mostly into UI/UX, Selecting high-quality and relevant resources for research was time-consuming. VueJS relies on javascript so being perfect in javascript is also necessary. It took time and was not possible for me to install VueJs application, so i could'nt run any code in my PC,So had to use online compile playgrounds. Filling 19 pages with the resources available was a bit difficult.

Conclusion

Ada, a statically-typed language, is renowned for reliability in safety-critical systems and embedded applications. Its procedural paradigm emphasizes modular design, structured programming, and strong error management through exception handling.
It faces challenges in finding recent resources, its applications in aerospace and defense highlight its strength in algorithmic design and real-time computing.

Vue.js, a popular JavaScript framework, employs an event-driven paradigm for reactive and interactive web interfaces. Vue.js features a declarative style, facilitating UI development through components and data binding. Its strength lies in building modern, dynamic web applications with real-time updates. Despite challenges in navigating the rapidly evolving JavaScript ecosystem, Vue.js is widely used in front-end development, providing a versatile solution for creating responsive single-page applications.
Both Ada and Vue.js demonstrate unique strengths, catering to distinct domains and reflecting the diverse landscape of programming paradigms.

References

- https://thenewstack.io/the-basics-of-event-driven-architectures/
- https://beginnersoftwaredeveloper.com/a-guide-to-vue-events-with-vue-event-examples/
- https://www.studysmarter.co.uk/explanations/computer-science/computer-programming/event-driven-programming/
- https://vuejsdevelopers.com/2020/01/06/handling-events-vue-js/
- https://innstelios.blogspot.com/
- https://thecloudstrap.com/chapter-1-ada-programming-language/OverviewofADAprogramminglangu
- https://piembsystech.com/introduction-to-ada-programming-language/